AHMET16 / **Tanzania-Water-Wells** Public

Code Issues Pull requests Actions Projects Wiki Security

main

Tanzania-Water-Wells / Tanzanian Water Wells .ipynb

AHMET16 Add files via upload History

1 contributor

9.71 MB

Tanzanian Water Wells Status Prediction

Overview

Tanzania is a developing country that struggles to get clean water to its population of 59 million people. According to WHO, 1 in 6 people in Tanzania lack access to safe drinking water and 29 million don't have access to improved sanitation. The focus of this project is to build a classification model to predict the functionality of waterpoints in Tanzania given data provided by Taarifa and the Tanzanian Ministry of Water. The model was built from a dataset containing information about the source of water and status of the waterpoint (functional, functional but needs repairs, and non functional) using an iterative approach and can be found here. The dataset contains 60,000 waterpoints in Tanzania and the following features:

Business Problem

The Tanzanian government has a severe water crisis on their hands as a result of the vast number of non functional wells and they have asked for help. They want to be able to predict the statuses of which pumps are functional, functional but need repair, and non functional in order to improve their maintenance operations and ensure that it's residents have access to safe drinking water. The data has been collected by and is provided by Taarifa and the Tanzanian Ministry of Water with the hope that the information provided by each waterpoint can aid understanding in which waterpoints will fail.

I have partnered with the Tanzanian government to build a classification model to predict the status of the waterpoints using the dataset provided. I will use the precision of the functional wells as my main metric for model selection, as a non functional well being predicted as a functional well would be more detrimental to their case, but will provide and discuss several metrics for each model.

Data Understanding

The dataset used for this analysis can be found here. It contains a wealth of information about waterpoints in Tanzania and the status of their operation. The target variable has 3 different options for it's status:

- functional - the waterpoint is operational and there are no repairs needed
- functional needs repair - the waterpoint is operational, but needs repairs
- non functional - the waterpoint is not operational Below I will import the dataset and start my investigation of relevant information it may contain. Let's get started!

started.

In [4]:
```
# Import standard packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, GridSearchCV, cross
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE, SMOTENC

# Classification Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
from sklearn.dummy import DummyClassifier
from xgboost.sklearn import XGBClassifier

from sklearn.metrics import plot_confusion_matrix, accuracy_score, f1_sco
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc, roc_auc_score

# Scalers
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, label_binarize

# Categorical Create Dummies
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import ExtraTreesClassifier
```

In [5]:
```
# Data Import Train Set
df_train_set = pd.read_csv('training_set_values.csv', index_col='id')
df_train_set
```

Out[5]:

	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitu
id							
69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.8563
8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.1474
34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.8213
67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.1552

19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.8253
...	...	...	...	...	...	...	
60739	10.0	2013-05-03	Germany Republi	1210	CES	37.169807	-3.2538
27263	4700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.249991	-9.0706
37057	0.0	2011-04-11	NaN	0	NaN	34.017087	-8.7504
31282	0.0	2011-03-08	Malec	0	Musa	35.861315	-6.3785
26348	0.0	2011-03-23	World Bank	191	World	38.104048	-6.7474

59400 rows × 39 columns

In [6]:

```
# Data import Training set labels
df_train_labels = pd.read_csv('training_set_labels.csv', index_col='id')
df_train_labels
```

Out[6]:

id	status_group
69572	functional
8776	functional
34310	functional
67743	non functional
19728	functional
...	...
60739	functional
27263	functional
37057	functional
31282	functional
26348	functional

59400 rows × 1 columns

In [7]:

```
#Merge datasets
df = pd.merge(df_train_labels, df_train_set, how = 'inner', on='id')
```

In [8]:

```
#Reset index
```

```
df.reset_index(inplace=True)
df.head()
```

Out[8]:

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	lo
0	69572	functional	6000.0	2011-03-14	Roman	1390	Roman	34
1	8776	functional	0.0	2013-03-06	Grumeti	1399	GRUMETI	34
2	34310	functional	25.0	2013-02-25	Lottery Club	686	World vision	37
3	67743	non functional	0.0	2013-01-28	Unicef	263	UNICEF	38
4	19728	functional	0.0	2011-07-13	Action In A	0	Artisan	31

5 rows × 41 columns

In [9]:

```
df['permit']
```

Out[9]:

```
0        False
1         True
2         True
3         True
4         True
         ...  
59395     True
59396     True
59397    False
59398     True
59399     True
Name: permit, Length: 59400, dtype: object
```

In [10]:

```
# Check datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   status_group           59400 non-null  object 
 2   amount_tsh             59400 non-null  float64
 3   date_recorded          59400 non-null  object 
 4   funder                 55765 non-null  object 
 5   gps_height             59400 non-null  int64  
 6   installer              55745 non-null  object 
 7   longitude              59400 non-null  float64
 8   latitude               59400 non-null  float64
 9   wpt_name               59400 non-null  object 
 10  num_private            59400 non-null  int64  
```

```
 11  basin                  59400 non-null  object
 12  subvillage             59029 non-null  object
 13  region                 59400 non-null  object
 14  region_code            59400 non-null  int64
 15  district_code          59400 non-null  int64
 16  lga                    59400 non-null  object
 17  ward                   59400 non-null  object
 18  population             59400 non-null  int64
 19  public_meeting         56066 non-null  object
 20  recorded_by            59400 non-null  object
 21  scheme_management      55523 non-null  object
 22  scheme_name            31234 non-null  object
 23  permit                 56344 non-null  object
 24  construction_year      59400 non-null  int64
 25  extraction_type        59400 non-null  object
 26  extraction_type_group  59400 non-null  object
 27  extraction_type_class  59400 non-null  object
 28  management             59400 non-null  object
 29  management_group       59400 non-null  object
 30  payment                59400 non-null  object
 31  payment_type           59400 non-null  object
 32  water_quality          59400 non-null  object
 33  quality_group          59400 non-null  object
 34  quantity               59400 non-null  object
 35  quantity_group         59400 non-null  object
 36  source                 59400 non-null  object
 37  source_type            59400 non-null  object
 38  source_class           59400 non-null  object
 39  waterpoint_type        59400 non-null  object
 40  waterpoint_type_group  59400 non-null  object
dtypes: float64(3), int64(7), object(31)
memory usage: 18.6+ MB
```

In [11]:

```
#Get stats on numeric columns
df.describe()
```

Out[11]:

	id	amount_tsh	gps_height	longitude	latitude	nu
count	59400.000000	59400.000000	59400.000000	59400.000000	5.940000e+04	594
mean	37115.131768	317.650385	668.297239	34.077427	-5.706033e+00	
std	21453.128371	2997.574558	693.116350	6.567432	2.946019e+00	
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01	
25%	18519.750000	0.000000	0.000000	33.090347	-8.540621e+00	
50%	37061.500000	0.000000	369.000000	34.908743	-5.021597e+00	
75%	55656.500000	20.000000	1319.250000	37.178387	-3.326156e+00	
max	74247.000000	350000.000000	2770.000000	40.345193	-2.000000e-08	17

In [12]:

```
#Check for duplicates
sum(df.duplicated())
```

Out[12]: 0

In [13]:

```
# Print all value counts to make observations
for col in df.columns:
    print(df[col].value_counts())
```

```
69572    1
27851    1
6924     1
61097    1
48517    1
        ..
59036    1
56446    1
3855     1
52786    1
26348    1
Name: id, Length: 59400, dtype: int64
functional                 32259
non functional             22824
functional needs repair     4317
Name: status_group, dtype: int64
0.0         41639
500.0        3102
50.0         2472
1000.0       1488
20.0         1463
            ...
6300.0          1
120000.0        1
138000.0        1
350000.0        1
59.0            1
Name: amount_tsh, Length: 98, dtype: int64
2011-03-15    572
2011-03-17    558
2013-02-03    546
2011-03-14    520
2011-03-16    513
             ...
2011-09-11      1
2011-08-31      1
2011-09-21      1
2011-08-30      1
2013-12-01      1
Name: date_recorded, Length: 356, dtype: int64
Government Of Tanzania    9084
Danida                    3114
Hesawa                    2202
Rwssp                     1374
World Bank                1349
                          ...
Rarymond Ekura               1
Justine Marwa                1
Municipal Council            1
Afdp                         1
Samlo                        1
Name: funder, Length: 1897, dtype: int64
 0      20438
-15        60
```

```
-16         55
-13         55
 1290       52
         ...
 2378        1
-54          1
 2057        1
 2332        1
 2366        1
Name: gps_height, Length: 2428, dtype: int64
DWE                17402
Government          1825
RWE                 1206
Commu               1060
DANIDA              1050
                     ...
Wizara  ya maji        1
TWESS                  1
Nasan workers          1
R                      1
SELEPTA                1
Name: installer, Length: 2145, dtype: int64
0.000000     1812
37.375717       2
38.340501       2
39.086183       2
33.005032       2
             ...
35.885754       1
36.626541       1
37.333530       1
38.970078       1
38.104048       1
Name: longitude, Length: 57516, dtype: int64
-2.000000e-08    1812
-6.985842e+00       2
-6.980220e+00       2
-2.476680e+00       2
-6.978263e+00       2
                 ...
-3.287619e+00       1
-8.234989e+00       1
-3.268579e+00       1
-1.146053e+01       1
-6.747464e+00       1
Name: latitude, Length: 57517, dtype: int64
none                        3563
Shuleni                     1748
Zahanati                     830
Msikitini                    535
Kanisani                     323
                            ...
Kwa Medadi                     1
Kwa Kubembeni                  1
Shule Ya Msingi Milanzi        1
Funua                          1
Kwa Mzee Lugawa                1
Name: wpt_name, Length: 37400, dtype: int64
0       58643
6          81
1          73
```

```
5          46
8          46
        ...
42          1
23          1
136         1
698         1
1402        1
Name: num_private, Length: 65, dtype: int64
Lake Victoria              10248
Pangani                     8940
Rufiji                      7976
Internal                    7785
Lake Tanganyika             6432
Wami / Ruvu                 5987
Lake Nyasa                  5085
Ruvuma / Southern Coast     4493
Lake Rukwa                  2454
Name: basin, dtype: int64
Madukani        508
Shuleni         506
Majengo         502
Kati            373
Mtakuja         262
               ...
Kipompo           1
Chanyamilima      1
Ikalime           1
Kemagaka          1
Kikatanyemba      1
Name: subvillage, Length: 19287, dtype: int64
Iringa           5294
Shinyanga        4982
Mbeya            4639
Kilimanjaro      4379
Morogoro         4006
Arusha           3350
Kagera           3316
Mwanza           3102
Kigoma           2816
Ruvuma           2640
Pwani            2635
Tanga            2547
Dodoma           2201
Singida          2093
Mara             1969
Tabora           1959
Rukwa            1808
Mtwara           1730
Manyara          1583
Lindi            1546
Dar es Salaam     805
Name: region, dtype: int64
11    5300
17    5011
12    4639
3     4379
5     4040
18    3324
19    3047
```

```
2     3024
16    2816
10    2640
4     2513
1     2201
13    2093
14    1979
20    1969
15    1808
6     1609
21    1583
80    1238
60    1025
90     917
7      805
99     423
9      390
24     326
8      300
40       1
Name: region_code, dtype: int64
1     12203
2     11173
3      9998
4      8999
5      4356
6      4074
7      3343
8      1043
30      995
33      874
53      745
43      505
13      391
23      293
63      195
62      109
60       63
0        23
80       12
67        6
Name: district_code, dtype: int64
Njombe          2503
Arusha Rural    1252
Moshi Rural     1251
Bariadi         1177
Rungwe          1106
                ...
Moshi Urban       79
Kigoma Urban      71
Arusha Urban      63
Lindi Urban       21
Nyamagana          1
Name: lga, Length: 125, dtype: int64
Igosi              307
Imalinyi           252
Siha Kati          232
Mdandu             231
Nduruma            217
                  ...
```

```
Uchindile           1
Thawi               1
Uwanja wa Ndege     1
Izia                1
Kinungu             1
Name: ward, Length: 2092, dtype: int64
0       21381
1        7025
200      1940
150      1892
250      1681
        ...
6330        1
5030        1
656         1
948         1
788         1
Name: population, Length: 1049, dtype: int64
True     51011
False     5055
Name: public_meeting, dtype: int64
GeoData Consultants Ltd    59400
Name: recorded_by, dtype: int64
VWC                 36793
WUG                  5206
Water authority      3153
WUA                  2883
Water Board          2748
Parastatal           1680
Private operator     1063
Company              1061
Other                 766
SWC                    97
Trust                  72
None                    1
Name: scheme_management, dtype: int64
K                        682
None                     644
Borehole                 546
Chalinze wate            405
M                        400
                        ...
Mradi wa maji Vijini       1
Villagers                  1
Magundi water supply       1
Saadani Chumv              1
Mtawanya                   1
Name: scheme_name, Length: 2696, dtype: int64
True     38852
False    17492
Name: permit, dtype: int64
0       20709
2010     2645
2008     2613
2009     2533
2000     2091
2007     1587
2006     1471
2003     1286
2011     1256
2004     1123
```

```
2012    1084
2002    1075
1978    1037
1995    1014
2005    1011
1999     979
1998     966
1990     954
1985     945
1980     811
1996     811
1984     779
1982     744
1994     738
1972     708
1974     676
1997     644
1992     640
1993     608
2001     540
1988     521
1983     488
1975     437
1986     434
1976     414
1970     411
1991     324
1989     316
1987     302
1981     238
1977     202
1979     192
1973     184
2013     176
1971     145
1960     102
1967      88
1963      85
1968      77
1969      59
1964      40
1962      30
1961      21
1965      19
1966      17
Name: construction_year, dtype: int64
gravity                      26780
nira/tanira                   8154
other                         6430
submersible                   4764
swn 80                        3670
mono                          2865
india mark ii                 2400
afridev                       1770
ksb                           1415
other - rope pump              451
other - swn 81                 229
windmill                       117
india mark iii                  98
cemo                            90
```

```
other - play pump               85
walimi                          48
climax                          32
other - mkulima/shinyanga        2
Name: extraction_type, dtype: int64
gravity            26780
nira/tanira         8154
other               6430
submersible         6179
swn 80              3670
mono                2865
india mark ii       2400
afridev             1770
rope pump            451
other handpump       364
other motorpump      122
wind-powered         117
india mark iii        98
Name: extraction_type_group, dtype: int64
gravity         26780
handpump        16456
other            6430
submersible      6179
motorpump        2987
rope pump         451
wind-powered      117
Name: extraction_type_class, dtype: int64
vwc                 40507
wug                  6515
water board          2933
wua                  2535
private operator     1971
parastatal           1768
water authority       904
other                 844
company               685
unknown               561
other - school         99
trust                  78
Name: management, dtype: int64
user-group    52490
commercial     3638
parastatal     1768
other           943
unknown         561
Name: management_group, dtype: int64
never pay                25348
pay per bucket            8985
pay monthly               8300
unknown                   8157
pay when scheme fails     3914
pay annually              3642
other                     1054
Name: payment, dtype: int64
never pay     25348
per bucket     8985
monthly        8300
unknown        8157
on failure     3914
annually       3642
other          1054
```

```
Other            1054
Name: payment_type, dtype: int64
soft                  50818
salty                  4856
unknown                1876
milky                   804
coloured                490
salty abandoned         339
fluoride                200
fluoride abandoned       17
Name: water_quality, dtype: int64
good        50818
salty        5195
unknown      1876
milky         804
colored       490
fluoride      217
Name: quality_group, dtype: int64
enough          33186
insufficient    15129
dry              6246
seasonal         4050
unknown           789
Name: quantity, dtype: int64
enough          33186
insufficient    15129
dry              6246
seasonal         4050
unknown           789
Name: quantity_group, dtype: int64
spring                  17021
shallow well            16824
machine dbh             11075
river                    9612
rainwater harvesting     2295
hand dtw                  874
lake                      765
dam                       656
other                     212
unknown                    66
Name: source, dtype: int64
spring                  17021
shallow well            16824
borehole                11949
river/lake              10377
rainwater harvesting     2295
dam                       656
other                     278
Name: source_type, dtype: int64
groundwater    45794
surface        13328
unknown          278
Name: source_class, dtype: int64
communal standpipe             28522
hand pump                      17488
other                           6380
communal standpipe multiple     6103
improved spring                  784
cattle trough                    116
dam                                7
Name: waterpoint_type, dtype: int64
```

```
communal standpipe    34625
hand pump             17488
other                  6380
improved spring         784
cattle trough           116
dam                       7
Name: waterpoint_type_group, dtype: int64
```

In [14]:

```
# Check null values
df.isna().sum()
```

Out[14]:

```
id                           0
status_group                 0
amount_tsh                   0
date_recorded                0
funder                    3635
gps_height                   0
installer                 3655
longitude                    0
latitude                     0
wpt_name                     0
num_private                  0
basin                        0
subvillage                 371
region                       0
region_code                  0
district_code                0
lga                          0
ward                         0
population                   0
public_meeting            3334
recorded_by                  0
scheme_management         3877
scheme_name              28166
permit                    3056
construction_year            0
extraction_type              0
extraction_type_group        0
extraction_type_class        0
management                   0
management_group             0
payment                      0
payment_type                 0
water_quality                0
quality_group                0
quantity                     0
quantity_group               0
source                       0
source_type                  0
source_class                 0
waterpoint_type              0
waterpoint_type_group        0
dtype: int64
```

In [15]:

```
# Check unique values for categorical data
obj_df = df.select_dtypes(include=['object'])
obj_df.nunique()
```

Out[15]:

```
status_group                 3
```

Out[15]:

```
date_recorded              356
funder                    1897
installer                 2145
wpt_name                 37400
basin                        9
subvillage               19287
region                      21
lga                        125
ward                      2092
public_meeting               2
recorded_by                  1
scheme_management           12
scheme_name               2696
permit                       2
extraction_type             18
extraction_type_group       13
extraction_type_class        7
management                  12
management_group             5
payment                      7
payment_type                 7
water_quality                8
quality_group                6
quantity                     5
quantity_group               5
source                      10
source_type                  7
source_class                 3
waterpoint_type              7
waterpoint_type_group        6
dtype: int64
```

Initial Observations

Missing Values

scheme_name has the most missing values, followed by funder, installer, public_meeting, scheme_management, and permit with ~3,000 null values, and then subvillage with 371 null values. Several of these columns will be deleted as they appear to duplicate other columns, and I will investigate installer, permit, and subvillage further.

Data types

wpt_name, subvillage, ward, scheme_name, installer, funder, and date_recorded are categorical features that have unique values in the thousands. This will be a problem with dummy variables, will likely remove or feature engineer. I will drop recorded_by as it has the same value for all rows. num_private is not defined on the DrivenData site, and it is not obvious what the feature indicates. id column will be dropped. public_meeting and permit are boolean. construction_year, latitude, longitude, gps_height, amount_tsh, and population all have thousands of rows of 0 entered. I will drop rows for most of these variables that have 0 entered, and will have to investigate further for real data on some columns.

Duplicate and Similar Data

The following columns all contain duplicate or similar data, will remove features that will cause multicollinearity:

- extraction_type, extraction_type_group, and extraction_type_class
- payment and payment_type
- water_quality and quality_group
- quanitity and quantity_group
- source and source_type
- waterpoint_type and waterpoint_type_group
- region and region_code ### Data Cleaning In this section, I will clean the dataset by removing similar and unnecessary columns and trim the dataset of remaining null values. I will also further investigate whether some columns contain the same information if it was not immediately obvious. There are several rows containing 0 enteries in some column information. I will investigate whether I believe the data to be real instead of a placeholder.

Drop duplicate and columns with similar information

I will keep extraction_type_class and remove extraction_type and extraction_type_group as it's columns values appear to be the most relevant for the project. scheme_name will be dropped for it's many null values. Other columns will be removed at this point due to irrelavancy, duplicates, null values, and some others will have to be investigated after the first drop.

In [16]:
```
# Columns to be dropped
dropped_columns = ['extraction_type', 'extraction_type_group', 'payment',
                   'quantity_group', 'source', 'waterpoint_type_group', '
                   'id', 'subvillage', 'wpt_name', 'ward', 'funder', 'dat
                   'region_code', 'district_code', 'lga', 'scheme_managem
```

In [17]:
```
df = df.drop(dropped_columns, axis=1)
```

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   status_group           59400 non-null  object
 1   amount_tsh             59400 non-null  float64
 2   gps_height             59400 non-null  int64
 3   installer              55745 non-null  object
```

```
 3   installer              55745 non-null  object
 4   longitude              59400 non-null  float64
 5   latitude               59400 non-null  float64
 6   basin                  59400 non-null  object
 7   region                 59400 non-null  object
 8   population             59400 non-null  int64
 9   permit                 56344 non-null  object
 10  construction_year      59400 non-null  int64
 11  extraction_type_class  59400 non-null  object
 12  management             59400 non-null  object
 13  management_group       59400 non-null  object
 14  payment_type           59400 non-null  object
 15  water_quality          59400 non-null  object
 16  quantity               59400 non-null  object
 17  source_type            59400 non-null  object
 18  waterpoint_type        59400 non-null  object
dtypes: float64(3), int64(3), object(13)
memory usage: 8.6+ MB
```

Dealing with null values

In [19]:

```
#Check for nulls
df.isna().sum()
```

Out[19]:

```
status_group                0
amount_tsh                  0
gps_height                  0
installer                3655
longitude                   0
latitude                    0
basin                       0
region                      0
population                  0
permit                   3056
construction_year           0
extraction_type_class       0
management                  0
management_group            0
payment_type                0
water_quality               0
quantity                    0
source_type                 0
waterpoint_type             0
dtype: int64
```

In [20]:

```
# Drop all remaining null values from our dataset
df = df.dropna()
```

In [21]:

```
#Check to see that it worked
df.isna().sum()
```

Out[21]:

```
status_group             0
amount_tsh               0
gps_height               0
installer                0
longitude                0
latitude                 0
```

```
basin                    0
region                   0
population               0
permit                   0
construction_year        0
extraction_type_class    0
management               0
management_group         0
payment_type             0
water_quality            0
quantity                 0
source_type              0
waterpoint_type          0
dtype: int64
```

In [22]:

```
# Convert boolean permit to integers
df['permit'] = df['permit'].astype(int)
```

In [23]:

```
# Check to see that it worked
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55102 entries, 0 to 59399
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           55102 non-null  object 
 1   amount_tsh             55102 non-null  float64
 2   gps_height             55102 non-null  int64  
 3   installer              55102 non-null  object 
 4   longitude              55102 non-null  float64
 5   latitude               55102 non-null  float64
 6   basin                  55102 non-null  object 
 7   region                 55102 non-null  object 
 8   population             55102 non-null  int64  
 9   permit                 55102 non-null  int64  
 10  construction_year      55102 non-null  int64  
 11  extraction_type_class  55102 non-null  object 
 12  management             55102 non-null  object 
 13  management_group       55102 non-null  object 
 14  payment_type           55102 non-null  object 
 15  water_quality          55102 non-null  object 
 16  quantity               55102 non-null  object 
 17  source_type            55102 non-null  object 
 18  waterpoint_type        55102 non-null  object 
dtypes: float64(3), int64(4), object(12)
memory usage: 8.4+ MB
```

Investigate management and management_group

I need to investigate these 2 columns further to see if they contain similar information.

In [24]:

```
df['management'].value_counts()
```

Out[24]:

```
vwc                     37416
```

```
wug                    6314
water board            2705
wua                    2307
private operator       1891
parastatal             1588
water authority         825
other                   733
company                 656
unknown                 491
other - school           99
trust                    77
Name: management, dtype: int64
```

In [25]:

```
df['management_group'].value_counts()
```

Out[25]:

```
user-group    48742
commercial     3449
parastatal     1588
other           832
unknown         491
Name: management_group, dtype: int64
```

The most data is contained in the user-group subcategory of management_group. I will groupby to investigate if the information is similar.

In [26]:

```
df.loc[df['management_group']=='user-group']['management'].value_counts()
```

Out[26]:

```
vwc            37416
wug             6314
water board     2705
wua             2307
Name: management, dtype: int64
```

The data is identical to the data contained in the management column in the subcategory of 'user-group'. I will drop management_group from our features.

In [27]:

```
#Drop column
df = df.drop('management_group', axis=1)
```

In [28]:

```
#Check to see that it worked
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55102 entries, 0 to 59399
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           55102 non-null  object 
 1   amount_tsh             55102 non-null  float64
 2   gps_height             55102 non-null  int64  
 3   installer              55102 non-null  object 
 4   longitude              55102 non-null  float64
 5   latitude               55102 non-null  float64
 6   basin                  55102 non-null  object 
 7   region                 55102 non-null  object 
 8   population             55102 non-null  int64
```

```
 8   population             55102 non-null  int64
 9   permit                 55102 non-null  int64
 10  construction_year      55102 non-null  int64
 11  extraction_type_class  55102 non-null  object
 12  management             55102 non-null  object
 13  payment_type           55102 non-null  object
 14  water_quality          55102 non-null  object
 15  quantity               55102 non-null  object
 16  source_type            55102 non-null  object
 17  waterpoint_type        55102 non-null  object
dtypes: float64(3), int64(4), object(11)
memory usage: 8.0+ MB
```

In [29]:

```python
for col in df.columns:
    print(df[col].value_counts())
```

```
functional                 29885
non functional             21381
functional needs repair     3836
Name: status_group, dtype: int64
0.0         37811
500.0        3071
50.0         2333
1000.0       1442
20.0         1427
            ...
53.0            1
138000.0        1
306.0           1
6300.0          1
59.0            1
Name: amount_tsh, Length: 95, dtype: int64
 0      18310
-15        54
-16        51
 303       51
-13        50
         ...
 2401       1
 2299       1
 2623       1
 2627       1
 2366       1
Name: gps_height, Length: 2426, dtype: int64
DWE               17361
Government         1788
RWE                1203
Commu              1060
DANIDA             1049
                  ...
B.A.P                 1
R                     1
Nasan workers         1
TWESS                 1
SELEPTA               1
Name: installer, Length: 2056, dtype: int64
0.000000     1793
33.005032       2
32.924886       2
32.993683       2
```

```
36.802490       2
              ...
32.904856       1
36.964268       1
34.736458       1
38.804318       1
38.104048       1
Name: longitude, Length: 53261, dtype: int64
-2.000000e-08    1793
-2.528716e+00       2
-6.958716e+00       2
-7.056923e+00       2
-2.515321e+00       2
                  ...
-2.068065e+00       1
-7.595047e+00       1
-9.645797e+00       1
-1.029702e+01       1
-6.747464e+00       1
Name: latitude, Length: 53263, dtype: int64
Lake Victoria              9705
Pangani                    8674
Rufiji                     7197
Internal                   6468
Lake Tanganyika            6406
Wami / Ruvu                5950
Ruvuma / Southern Coast    4481
Lake Nyasa                 3769
Lake Rukwa                 2452
Name: basin, dtype: int64
Iringa           5285
Shinyanga        4940
Kilimanjaro      4237
Morogoro         3995
Kagera           3224
Mwanza           3050
Arusha           2898
Kigoma           2805
Mbeya            2703
Ruvuma           2636
Tanga            2546
Pwani            2497
Dodoma           2199
Tabora           1942
Rukwa            1805
Mtwara           1725
Mara             1592
Manyara          1580
Lindi            1542
Singida          1124
Dar es Salaam     777
Name: region, dtype: int64
0       19250
1        6100
150      1854
200      1815
250      1605
         ...
723         1
5050        1
408         1
```

```
1885        1
788         1
Name: population, Length: 1026, dtype: int64
1    38195
0    16907
Name: permit, dtype: int64
0       18392
2008     2568
2009     2490
2010     2427
2000     1565
2007     1557
2006     1447
2003     1276
2011     1211
2004     1107
2002     1064
1978     1027
2012     1025
2005      983
1995      978
1999      950
1985      941
1998      921
1984      777
1996      766
1982      741
1972      705
1994      703
1974      675
1990      666
1980      647
1992      632
1997      612
1993      595
2001      530
1988      520
1983      487
1975      437
1986      431
1976      411
1991      322
1989      316
1970      310
1987      297
1981      237
1977      199
1979      192
1973      183
2013      173
1971      145
1963       84
1967       83
1968       68
1969       59
1960       45
1964       40
1962       29
1961       20
1965       19
```

```
1966       17
Name: construction_year, dtype: int64
gravity         24439
handpump        15779
other            5983
submersible      5759
motorpump        2689
rope pump         348
wind-powered      105
Name: extraction_type_class, dtype: int64
vwc                 37416
wug                  6314
water board          2705
wua                  2307
private operator     1891
parastatal           1588
water authority       825
other                 733
company               656
unknown               491
other - school         99
trust                  77
Name: management, dtype: int64
never pay     23097
per bucket     8666
monthly        8034
unknown        7021
on failure     3773
annually       3521
other           990
Name: payment_type, dtype: int64
soft                  47474
salty                  4652
unknown                1279
milky                   785
coloured                391
salty abandoned         329
fluoride                175
fluoride abandoned       17
Name: water_quality, dtype: int64
enough          31664
insufficient    13695
dry              5768
seasonal         3344
unknown           631
Name: quantity, dtype: int64
shallow well            16073
spring                  15792
borehole                10954
river/lake               9430
rainwater harvesting     1978
dam                       629
other                     246
Name: source_type, dtype: int64
communal standpipe             25551
hand pump                      16698
communal standpipe multiple     6012
other                           6004
improved spring                  745
cattle trough                     86
```

```
dam                                   6
Name: waterpoint_type, dtype: int64
```

After our first round of cleaning, there are several features we need to examine further:

- status_group is an unbalanced target, may need to look into further during modeling and apply SMOTE.
- There are several columns with thousands of 0 entries - amount_tsh, gps_height, longitude, latitude, population, construction_year.

Construction year

In [30]:

```
df['construction_year'].value_counts()
```

Out[30]:

```
0       18392
2008     2568
2009     2490
2010     2427
2000     1565
2007     1557
2006     1447
2003     1276
2011     1211
2004     1107
2002     1064
1978     1027
2012     1025
2005      983
1995      978
1999      950
1985      941
1998      921
1984      777
1996      766
1982      741
1972      705
1994      703
1974      675
1990      666
1980      647
1992      632
1997      612
1993      595
2001      530
1988      520
1983      487
1975      437
1986      431
1976      411
1991      322
1989      316
1970      310
1987      297
1981      237
1977      199
1979      192
```

```
1973     183
2013     173
1971     145
1963      84
1967      83
1968      68
1969      59
1960      45
1964      40
1962      29
1961      20
1965      19
1966      17
Name: construction_year, dtype: int64
```

In [31]:

```
# Finding mean and median without zero values
df.loc[df['construction_year']!=0].describe()
```

Out[31]:

	amount_tsh	gps_height	longitude	latitude	population	
count	36710.000000	36710.000000	36710.000000	36710.000000	36710.000000	3671C
mean	471.881843	982.395015	36.015003	-6.358975	268.881694	(
std	3074.841656	623.784917	2.609370	2.762486	542.812926	C
min	0.000000	-63.000000	29.607122	-11.649440	0.000000	C
25%	0.000000	351.000000	34.671850	-8.855908	30.000000	C
50%	0.000000	1116.500000	36.691907	-6.351197	150.000000	1
75%	200.000000	1471.000000	37.896261	-3.731978	304.000000	1
max	250000.000000	2770.000000	40.345193	-1.042375	30500.000000	1

In [32]:

```
#Replace 0 values in construction_year with 1950 to aid visualization
df['construction_year'].replace(to_replace = 0, value = 1950, inplace=Tru
```

In [33]:

```
#Check to see if it worked
df['constuction_year'].value_counts()
```

Out[33]:

```
1950    18392
2008     2568
2009     2490
2010     2427
2000     1565
2007     1557
2006     1447
2003     1276
2011     1211
2004     1107
2002     1064
1978     1027
2012     1025
2005      983
1995      978
```

```
1999     950
1985     941
1998     921
1984     777
1996     766
1982     741
1972     705
1994     703
1974     675
1990     666
1980     647
1992     632
1997     612
1993     595
2001     530
1988     520
1983     487
1975     437
1986     431
1976     411
1991     322
1989     316
1970     310
1987     297
1981     237
1977     199
1979     192
1973     183
2013     173
1971     145
1963      84
1967      83
1968      68
1969      59
1960      45
1964      40
1962      29
1961      20
1965      19
1966      17
Name: construction_year, dtype: int64
```

It is unfortunate that there are 19,000 entries with 0 for the construction_year. These may be natural and spring fed sources that were never "constructed". I chose to replace the 0 values with 1950, so they are still the "oldest" in the dataset, but will aid in visualizing the functionality of the pumps by the year they were made.

Latitude/Longitude zeros

In [34]:

```
df.longitude.value_counts()
```

Out[34]:

```
0.000000     1793
33.005032       2
32.924886       2
32.993683       2
36.802490       2
             ...
```

```
32.904856       1
36.964268       1
34.736458       1
38.804318       1
38.104048       1
Name: longitude, Length: 53261, dtype: int64
```

In [35]:

```
# Investigate longitude entries that are 0
df.loc[df['longitude'] == 0]
```

Out[35]:

	status_group	amount_tsh	gps_height	installer	longitude	latitude	ba
21	functional	0.0	0	DWE	0.0	-2.000000e-08	L Vict
53	non functional	0.0	0	Government	0.0	-2.000000e-08	L Vict
168	functional	0.0	0	WVT	0.0	-2.000000e-08	L Vict
177	non functional	0.0	0	DWE	0.0	-2.000000e-08	L Vict
253	functional needs repair	0.0	0	DWE	0.0	-2.000000e-08	L Vict
...	...	...	...	...	...	...	
59189	functional needs repair	0.0	0	DWE	0.0	-2.000000e-08	L Vict
59208	functional	0.0	0	DWE	0.0	-2.000000e-08	L Vict
59295	functional needs repair	0.0	0	DWE	0.0	-2.000000e-08	L Vict
59324	functional	0.0	0	World Vision	0.0	-2.000000e-08	L Vict
59374	functional	0.0	0	DWE	0.0	-2.000000e-08	L Vict

1793 rows × 18 columns

The 0s that are entered into the longitude column are also 0s in gps_height and -2e8 for latitude columns. I will drop these values from the dataset.

In [36]:

```
# Drop rows with 0 entered in longitude column
df = df.loc[df['longitude'] != 0]
```

In [37]:

```
# Check to see if it worked
df.describe()
```

Out[37]:

	amount_tsh	gps_height	longitude	latitude	population

count	53309.000000	53309.000000	53309.000000	53309.000000	53309.000000	533C
mean	337.580181	692.509670	35.186804	-5.849440	188.814515	
std	2714.547122	691.264883	2.670974	2.806529	474.147131	
min	0.000000	-90.000000	29.607122	-11.649440	0.000000	
25%	0.000000	0.000000	33.167340	-8.441371	0.000000	
50%	0.000000	438.000000	35.295878	-5.144420	45.000000	
75%	40.000000	1322.000000	37.353028	-3.359390	240.000000	
max	250000.000000	2770.000000	40.345193	-0.998464	30500.000000	

In [38]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53309 entries, 0 to 59399
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           53309 non-null  object 
 1   amount_tsh             53309 non-null  float64
 2   gps_height             53309 non-null  int64  
 3   installer              53309 non-null  object 
 4   longitude              53309 non-null  float64
 5   latitude               53309 non-null  float64
 6   basin                  53309 non-null  object 
 7   region                 53309 non-null  object 
 8   population             53309 non-null  int64  
 9   permit                 53309 non-null  int64  
 10  construction_year      53309 non-null  int64  
 11  extraction_type_class  53309 non-null  object 
 12  management             53309 non-null  object 
 13  payment_type           53309 non-null  object 
 14  water_quality          53309 non-null  object 
 15  quantity               53309 non-null  object 
 16  source_type            53309 non-null  object 
 17  waterpoint_type        53309 non-null  object 
dtypes: float64(3), int64(4), object(11)
memory usage: 7.7+ MB
```

In [39]:

```
df['installer']
```

Out[39]:

```
0               Roman
1             GRUMETI
2        World vision
3              UNICEF
4             Artisan
             ...     
59394        ML appro
59395             CES
59396            Cefa
59398            Musa
59399           World
Name: installer, Length: 53309, dtype: object
```

Looks like it all worked! I believe the amount_tsh and population 0 values are real so I will leave all data as is for vanilla models.

Installer - Several different spellings for same installer

In [40]:

```
#Check unique values after inital cleaning
df.nunique()
```

Out[40]:

```
status_group                3
amount_tsh                 95
gps_height               2426
installer                2024
longitude               53260
latitude                53262
basin                       9
region                     21
population               1026
permit                      2
construction_year          55
extraction_type_class       7
management                 12
payment_type                7
water_quality               8
quantity                    5
source_type                 7
waterpoint_type             7
dtype: int64
```

Upon checking the unique values for our categorical variables after trimming the dataset, installer still has 2024 unique entries, which will be a problem when we create dummies. We will need to cut down the amount of unique entries to not overload our model.

In [41]:

```
#Investigate 2024 unique values for installer
# pd.set_option("display.max_rows", None)
df['installer'].value_counts()
```

Out[41]:

```
DWE               16214
Government         1633
RWE                1178
Commu              1060
DANIDA             1049
                  ...
Centra govt           1
HESAWZ                1
CONCE                 1
B.A.P                 1
SELEPTA               1
Name: installer, Length: 2024, dtype: int64
```

There are several entries with typos and different variations of the same installer. I will attempt to fix some of the clerical errors and narrow down the amount of unique identifiers we will use for our model.

In [42]:

```
# Correct variations and misspellings in the installer column
df['installer'] = df['installer'].replace(to_replace = ('Central governme
                                          'Cental Government','Tanzania g
                                          'Centra Government', 'central g
                                          'TANZANIA GOVERNMENT','Central 
                                          'Tanzanian Government', 'Tanzar

df['installer'] = df['installer'].replace(to_replace = ('District COUNCIL
                                          'Counc','District council','Dis
                                          'Council', 'COUN', 'Distri', 'D
                                          value = 'District Council')

df['installer'] = df['installer'].replace(to_replace = ('villigers', 'vil
                                          'Villi', 'Village Council', 'Vi
                                          'Village community', 'Villaers'
                                          'Villege Council', 'Village cou
                                          'Villager', 'Village Technician
                                          'Village community members', 'V
                                          'Village govt', 'VILLAGERS', 'V

df['installer'] = df['installer'].replace(to_replace = ('District Water D
                                          'Distric Water Department'), va

df['installer'] = df['installer'].replace(to_replace = ('FinW', 'Fini wat
                                          'Finwater', 'FINN WATER', 'FinW
                                          value ='Fini Water')

df['installer'] = df['installer'].replace(to_replace = ('RC CHURCH', 'RC 
                                          'RC church', 'RC CATHORIC', 'Ch

df['installer'] = df['installer'].replace(to_replace = ('world vision', '
                                          'WORLD VISION', 'World Vission'

df['installer'] = df['installer'].replace(to_replace = ('Unisef','Unicef'

df['installer'] = df['installer'].replace(to_replace = 'DANID', value ='D

df['installer'] = df['installer'].replace(to_replace =('Commu', 'Communit
                                          'Adra /Community', 'Communit', 
                                          value ='Community')

df['installer'] = df['installer'].replace(to_replace = ('GOVERNMENT', 'GO
                                          'Gover', 'Gove', 'Governme', 'G

df['installer'] = df['installer'].replace(to_replace = ('Hesawa', 'hesawa

df['installer'] = df['installer'].replace(to_replace = ('JAICA', 'JICA', 
                                          value ='Jaica')
```

In [43]:

```
df['installer'] = df['installer'].replace(to_replace = ('KKKT _ Konde and
                                          value ='KKKT')

df['installer'] = df['installer'].replace(to_replace = '0', value ='Unkno
```

In [44]:

```
df['installer'].value_counts().head(20)
```

Out[44]:

```
DWE                    16214
```

```
Government              2468
Community               1791
DANIDA                  1601
HESAWA                  1180
RWE                     1178
District Council        1173
Central Government      1115
KKKT                    1102
Fini Water               952
Unknown                  780
TCRS                     702
World Vision             660
CES                      610
RC Church                484
Villagers                482
LGA                      408
WEDECO                   397
TASAF                    371
Jaica                    358
Name: installer, dtype: int64
```

Reduce Dimensionality for Installer

In [45]:

```
# Keep only top 20 installers as unique values
installer_20 = df.installer.value_counts(normalize=True).head(20).index.t


df['installer'] = [type_ if type_ in installer_20
                      else "OTHER" for type_ in df['installer']]
```

In [46]:

```
df.installer.value_counts()
```

Out[46]:

```
OTHER                  19283
DWE                    16214
Government              2468
Community               1791
DANIDA                  1601
HESAWA                  1180
RWE                     1178
District Council        1173
Central Government      1115
KKKT                    1102
Fini Water               952
Unknown                  780
TCRS                     702
World Vision             660
CES                      610
RC Church                484
Villagers                482
LGA                      408
WEDECO                   397
TASAF                    371
Jaica                    358
Name: installer, dtype: int64
```

In [47]:

```
###df.sort_values('installer', inplace= True)
```

In [48]:

```
##df
```

To reduce the dimensionality of the dataset, I made an "Other" category for installer if they were not in the top 20 installers of the dataset.

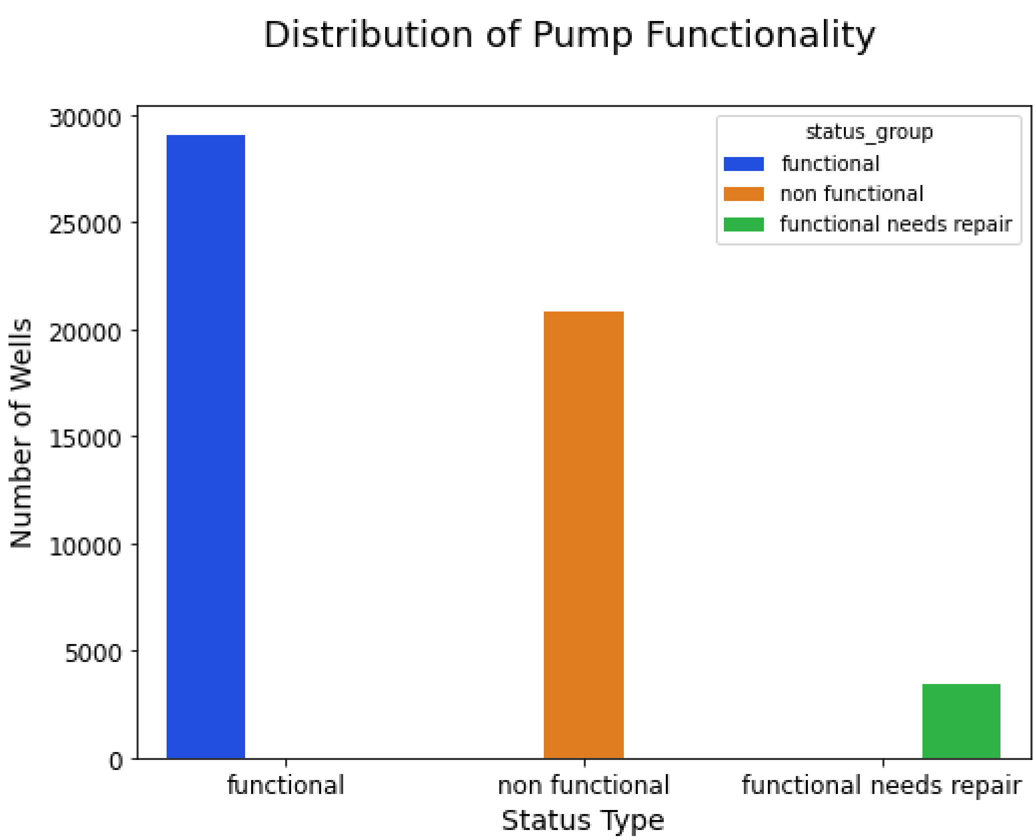
Modified Features Exploration

Column EDA

Target Feature Distribution

In [49]:

```
fig, ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='status_group', hue="status_group", palette='bright'

fig.suptitle('Distribution of Pump Functionality', fontsize=18)
plt.xlabel("Status Type", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.tick_params(labelsize='large')
plt.show()

fig.savefig('/Users/karaoglan/Desktop.jpeg');
```



In [50]:

```
df['status_group'].value_counts()
```

Out[50]:

```
functional                 29026
non functional             20829
```

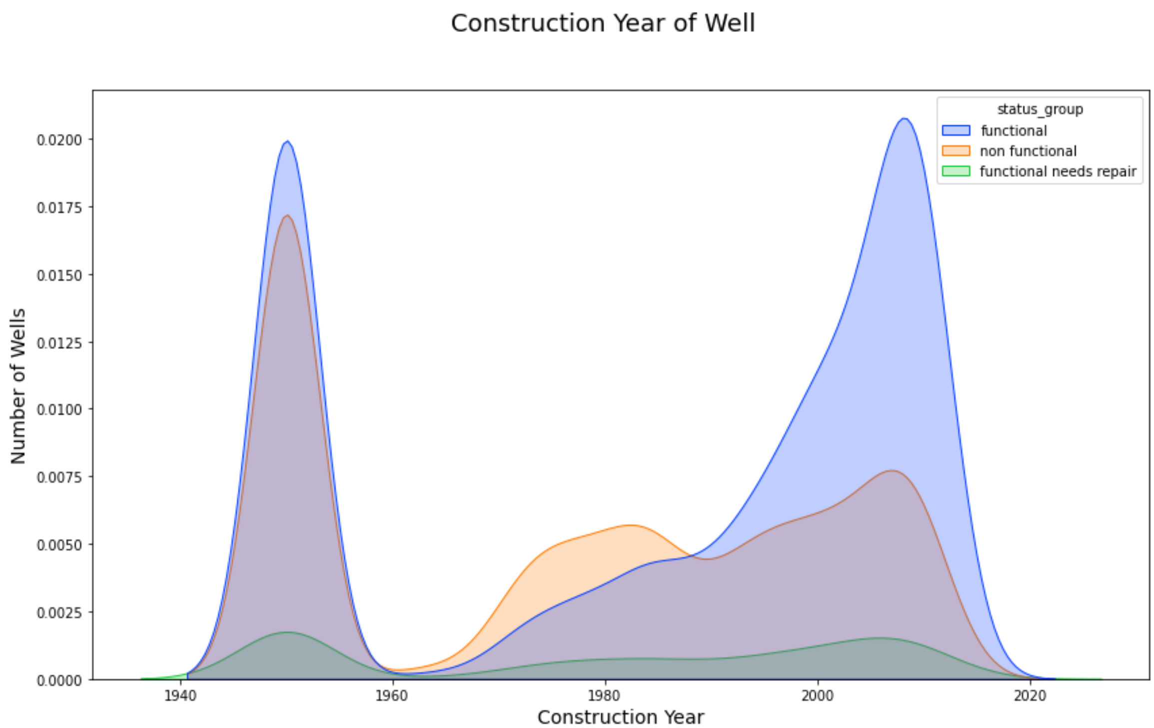
```
functional needs repair     3454
Name: status_group, dtype: int64
```

We have the most functional wells at ~29,000, followed by non functional wells at ~21,000, and the minority class, functional needs repair at ~3,500.

Construction year

In [51]:

```
fig, ax = plt.subplots(figsize=(14,8))
ax = sns.kdeplot(data=df, x='construction_year', hue='status_group', pale
fig.suptitle('Construction Year of Well', fontsize=18)
plt.xlabel("Construction Year", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show();
```



There is large amount of data in the year 1950 which were entered as 0 in the dataset, these may be natural sources and our distribution is normal for these sources. However, we can see the correlation of an older pump being more likely to be non functional and more functional newer pumps.

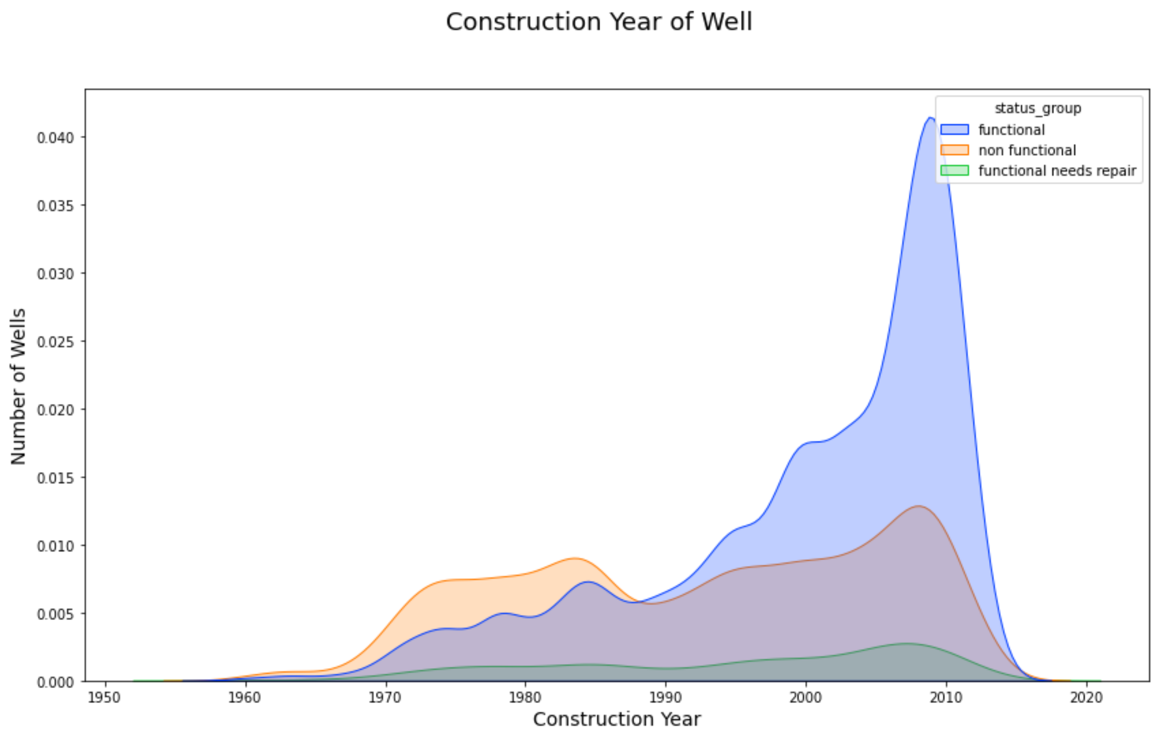
In [52]:

```
# Create df without entries in year 1950
const_year_df = df.loc[df['construction_year'] != 1950]
```

In [53]:

```
# Save without waterpoints from year 1950
fig, ax = plt.subplots(figsize=(14,8))
ax = sns.kdeplot(data=const_year_df, x='construction_year', hue='status_g
fig.suptitle('Construction Year of Well', fontsize=18)
plt.xlabel("Construction Year", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show()
```

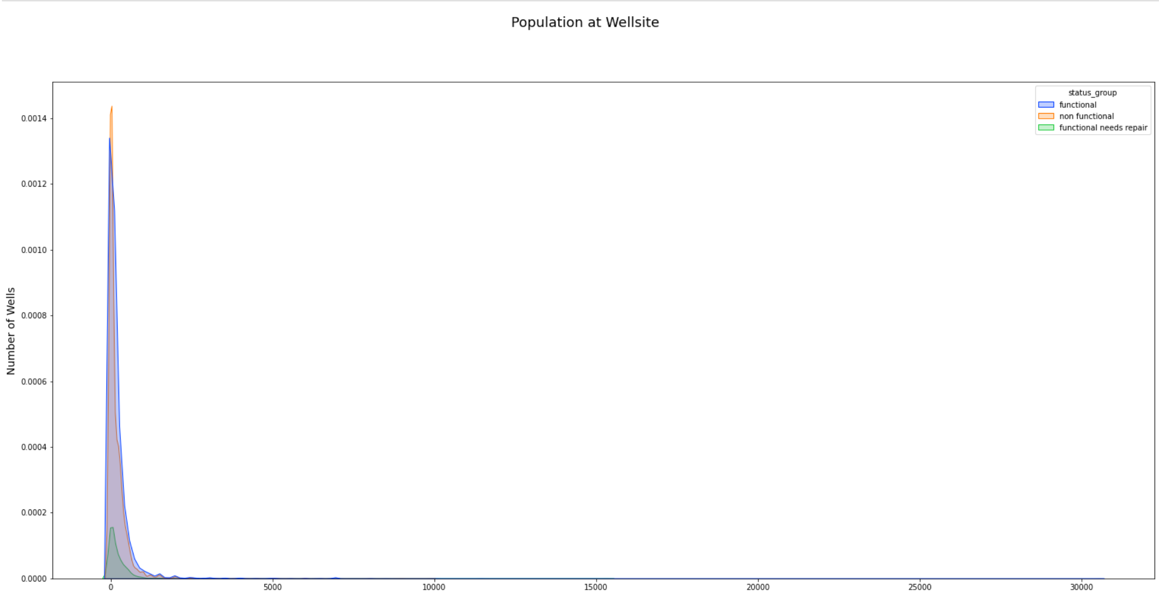
```
fig.savefig('/Users/karaoglan/Desktop.jpeg');
```



There are more non functional pumps than functional if they were built before 1988, but the rate of functionality keeps increasing after 1988.

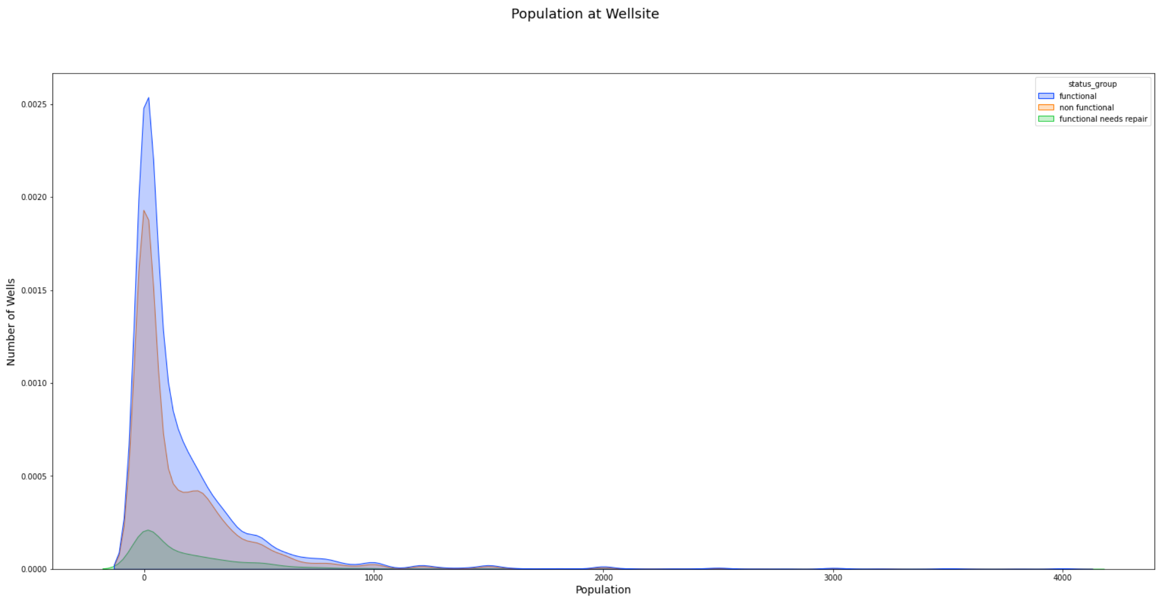
Population

In [54]:
```
fig, ax = plt.subplots(figsize=(26,12))
ax = sns.kdeplot(data=df, x='population', hue='status_group', palette='br
fig.suptitle('Population at Wellsite', fontsize=18)
plt.xlabel("Population", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show()
```



In [55]:

```
# Get a closer look
pop_df = df.loc[df['population'] <= 4000]

fig, ax = plt.subplots(figsize=(26,12))
ax = sns.kdeplot(data=pop_df, x='population', hue='status_group', palette
fig.suptitle('Population at Wellsite', fontsize=18)
plt.xlabel("Population", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show()
```



In [56]:

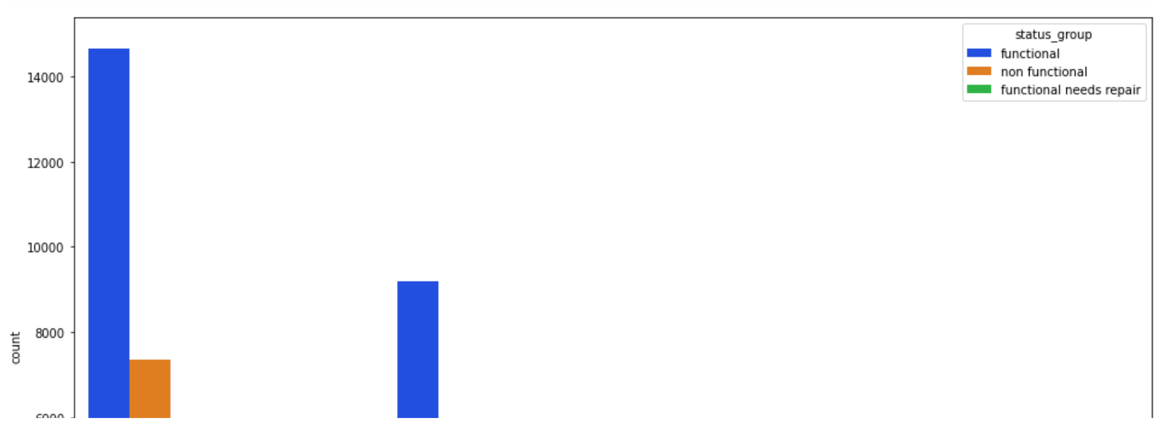
```
#df.sort_values[by= "functional", axis=0)
```

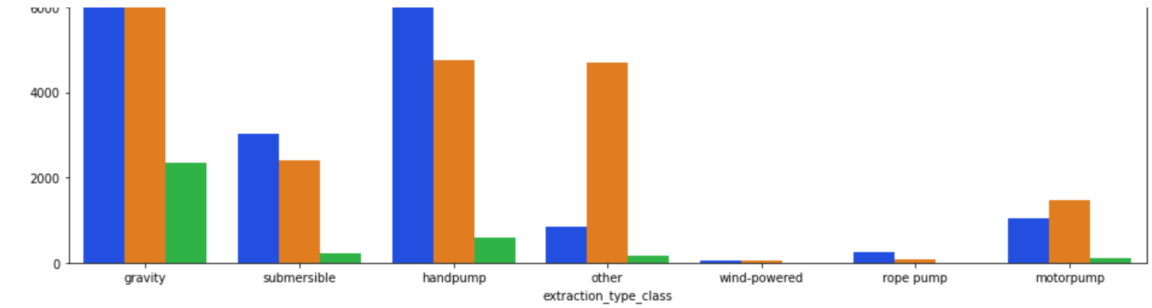
Overall, the distribution of pump functionality is similar across all population ranges and there isn't a lot of separation, with there being more functional wells than any other class. There isn't too much to draw from these graphs about population and functionality.

Extraction_type_class

In [57]:

```
plt.figure(figsize=(16,10))
ax = sns.countplot(x='extraction_type_class', hue='status_group', palette
```



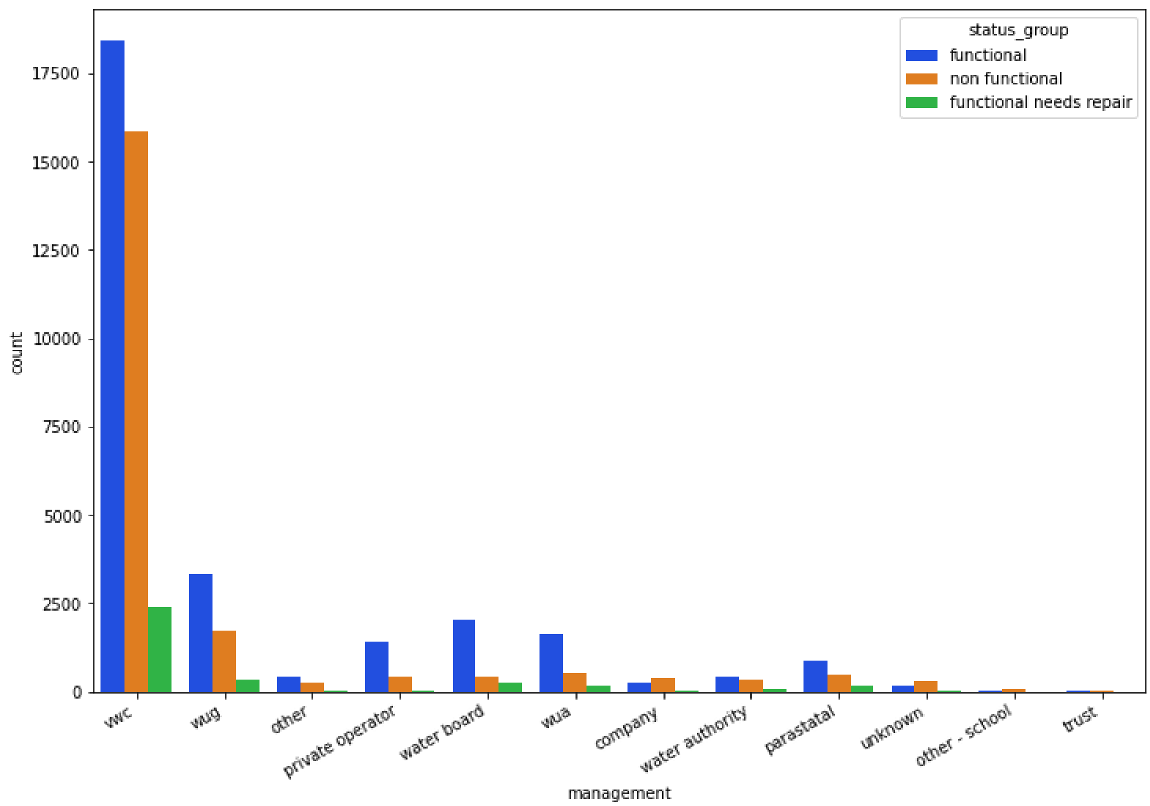


Other type and motorpump are especially non functioning. Gravity and handpump are the 2 largest types, and both have more functioning, but half non functioning.

Management

In [58]:

```
plt.figure(figsize=(12,8))
ax = sns.countplot(x='management', hue='status_group', palette='bright',
plt.xticks(rotation=30, ha='right');
```



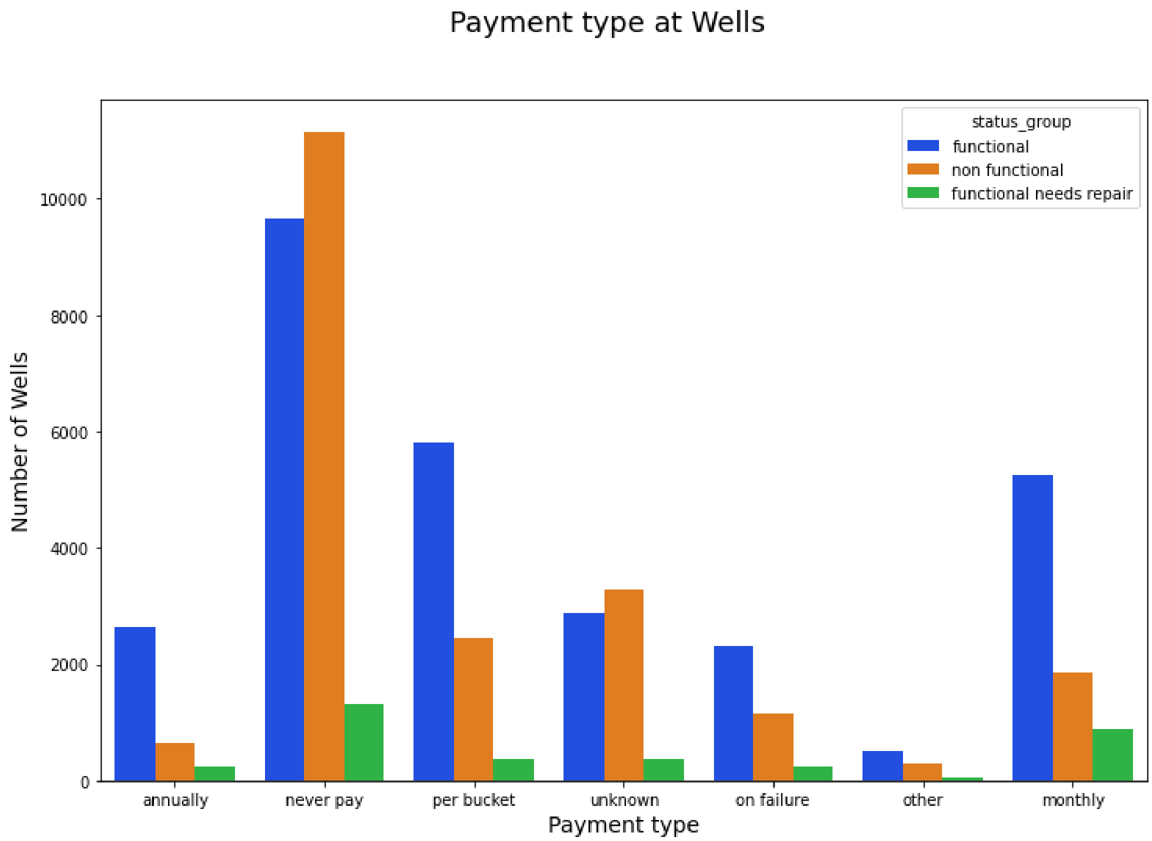
water board, wua, and private operators have a high rate of functionality.

Payment_type

In [59]:

```
fig, ax = plt.subplots(figsize=(12,8))
ax = sns.countplot(x='payment_type', hue="status_group", palette='bright'

fig.suptitle('Payment type at Wells', fontsize=18)
plt.xlabel("Payment type", fontsize=14)
```

```
plt.ylabel("Number of Wells", fontsize=14)
plt.show()

fig.savefig('/Users/karaoglan/Desktop.jpeg');
```

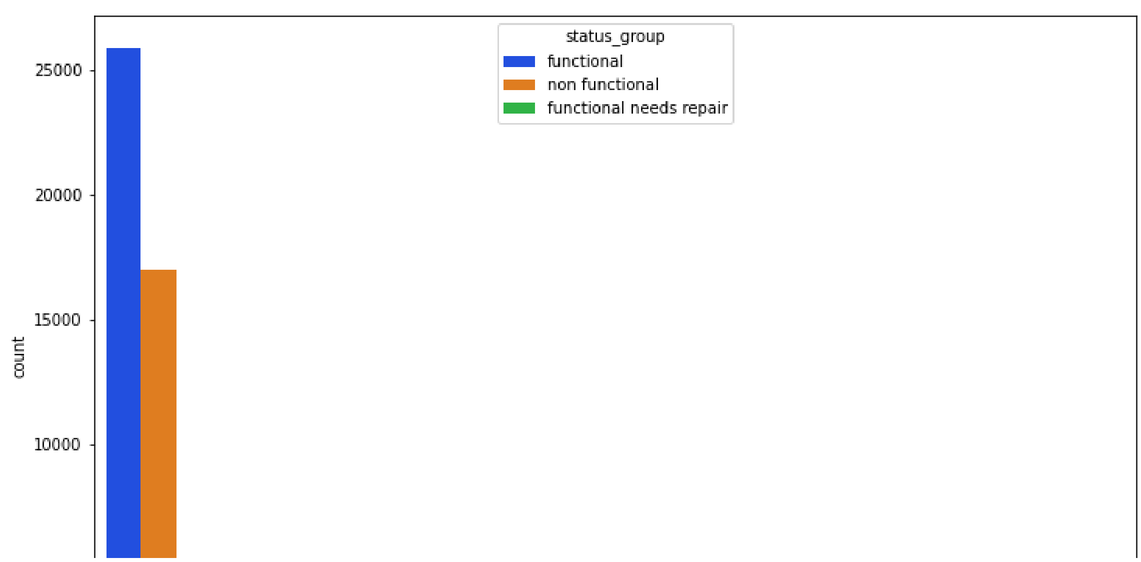


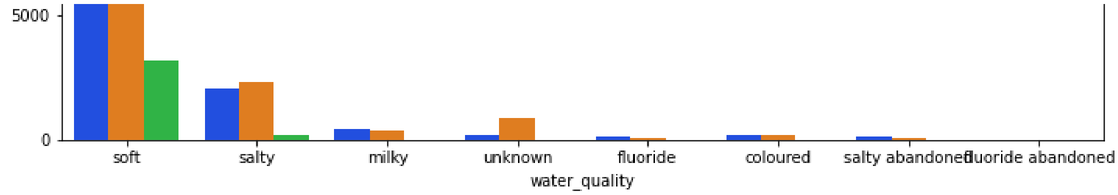
Never pay pumps have more non functioning waterpoints than functioning waterpoints. Some form of payment increases the functionality of the waterpoints.

Water quality

In [60]:

```
plt.figure(figsize=(12,8))
ax = sns.countplot(x='water_quality', hue='status_group', palette='bright
```



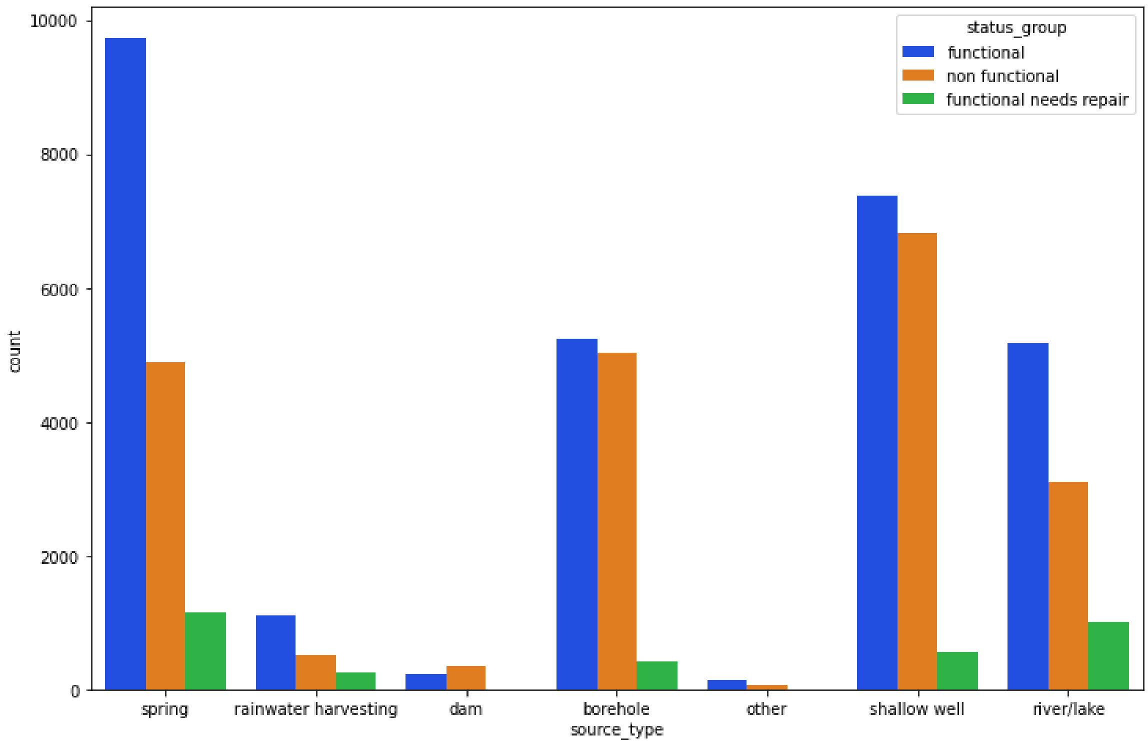


Soft water quality has a high rate of functional waterpoints, salty has a high rate of non functional waterpoints.

Source type

In [61]:

```
plt.figure(figsize=(12,8))
ax = sns.countplot(x='source_type', hue='status_group', palette='bright',
```

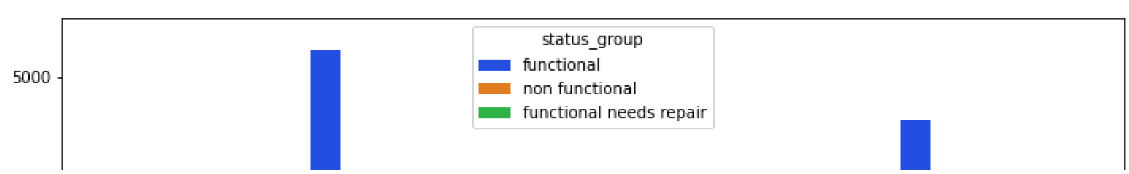


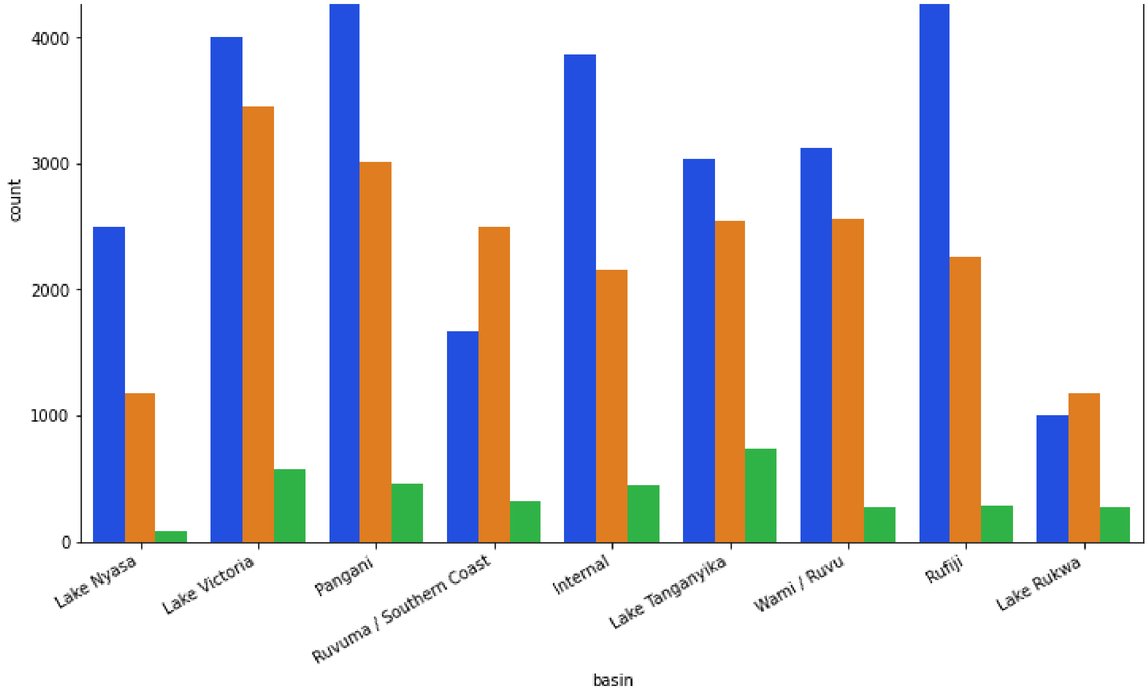
Even distribution of functional and nonfunctional boreholes. Many more functional springs and rivers than non functional.

Basin

In [62]:

```
plt.figure(figsize=(12,8))
ax = sns.countplot(x='basin', hue='status_group', palette='bright', data=
plt.xticks(rotation=30, ha='right');
```

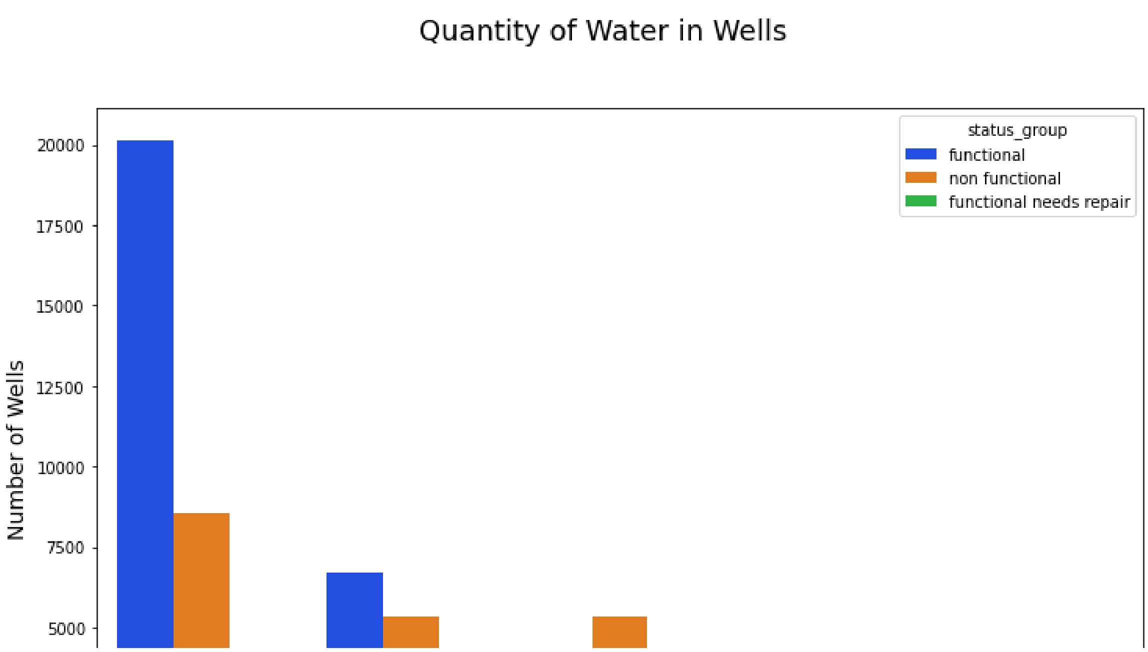


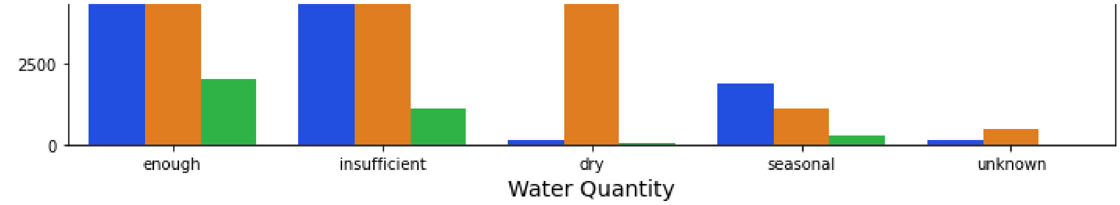


The Ruvuma/Southern Coast and Lake Rukwa basins have more non functioning wells than functional.

Quantity

In [63]:
```
fig, ax = plt.subplots(figsize=(12,8))
ax = sns.countplot(x='quantity', hue="status_group", palette='bright', da

fig.suptitle('Quantity of Water in Wells', fontsize=18)
plt.xlabel("Water Quantity", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show()

fig.savefig('/Users/karaoglan/Desktop.jpeg');
```

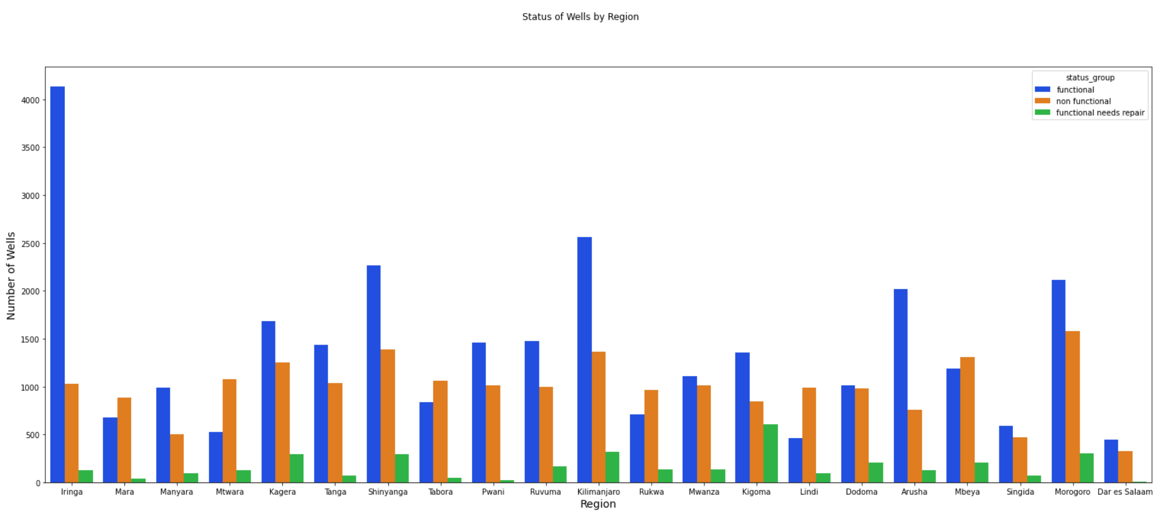




Dry waterpoints have a high chance of being non functional, as expected. If the waterpoint has enough water, there is a high chance of functionality.

Region

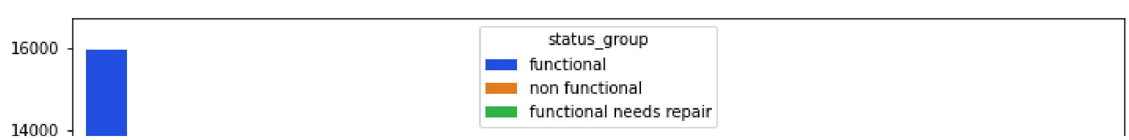
In [64]:
```
fig, ax = plt.subplots(figsize=(26,10))
ax = sns.countplot(x='region', hue="status_group", palette='bright', data

fig.suptitle('Status of Wells by Region')
plt.xlabel("Region", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show()

fig.savefig('/Users/karaoglan/Desktop.jpeg');
```

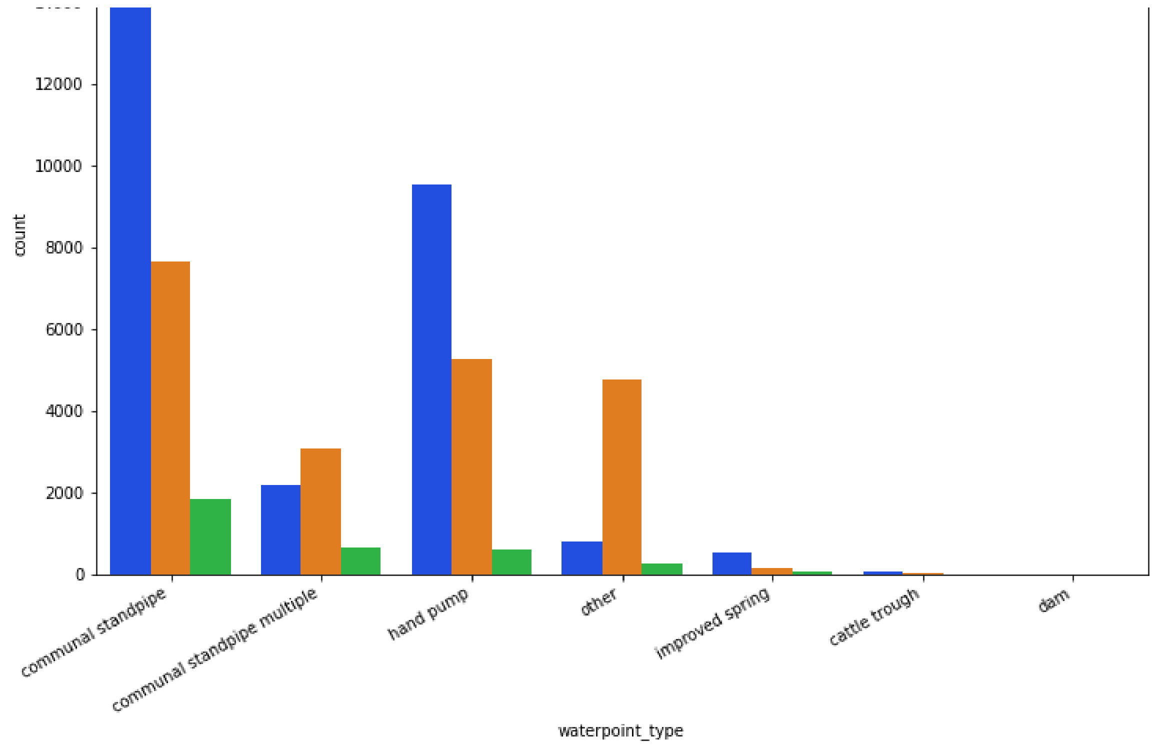


The Iringa region has a very high rate of functioning wells, followed by Kilimanjaro, Arusha, and Shinyanga. The worst regions for well perfomance are Mtwara, Mara, Rukwa, and Lindi.

Waterpoint type

In [65]:
```
plt.figure(figsize=(12,8))
ax = sns.countplot(x='waterpoint_type', hue='status_group', palette='brig
plt.xticks(rotation=30, ha='right');
```



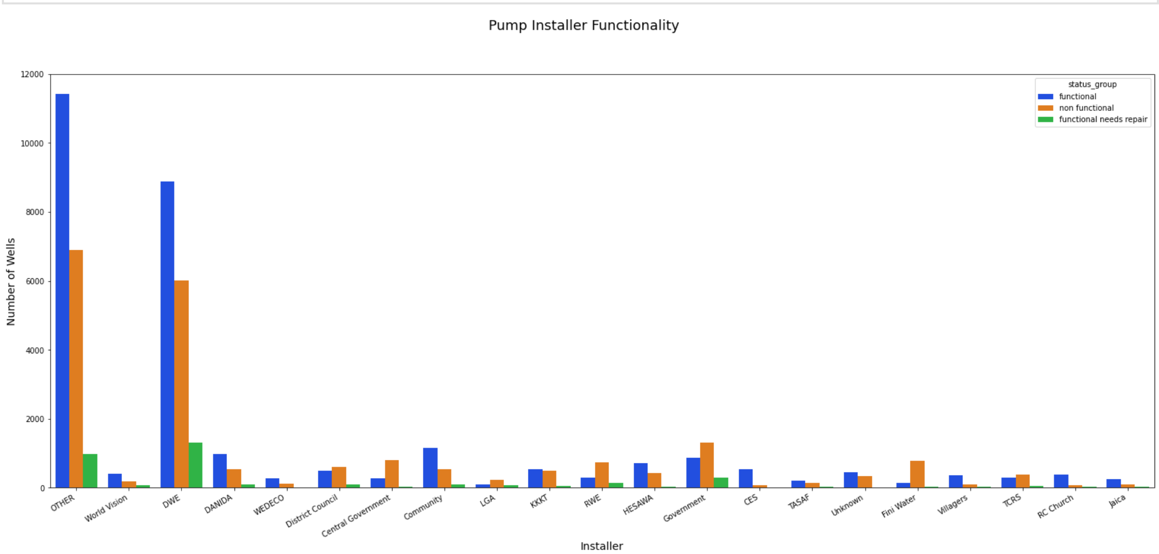


other and commuanal standpipe multiple have the highest rate of being non functioning

Installer

In [66]:

```
fig, ax = plt.subplots(figsize=(26,10))
ax = sns.countplot(x='installer', hue="status_group", palette='bright', c

fig.suptitle('Pump Installer Functionality', fontsize=18)
plt.xlabel("Installer", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.xticks(rotation=30, ha='right')
plt.show()

fig.savefig('/Users/karaoglan/Desktop.jpeg');
```



The government, Fini Water, RWE, and Distict Council have a high rate of non functioning wells. Other is out largest category.

Well Function map

In [67]:

```
import folium
from folium.plugins import FloatImage
```

In [68]:

```
# Create 3 dataframes for each status_group
df_f = df[df['status_group'] == 'functional']
df_nf = df[df['status_group'] == 'non functional']
df_fnr = df[df['status_group'] == 'functional needs repair']
```

In [69]:

```
# Create lists of latitude and longitude values
lat_f = [x for x in df_f['latitude']]
long_f = [x for x in df_f['longitude']]

lat_nf = [x for x in df_nf['latitude']]
long_nf = [x for x in df_nf['longitude']]

lat_fnr = [x for x in df_fnr['latitude']]
long_fnr = [x for x in df_fnr['longitude']]

lat_long_f = [(lat_f[i], long_f[i]) for i in range(len(lat_f))]
lat_long_nf = [(lat_nf[i], long_nf[i]) for i in range(len(lat_nf))]
lat_long_fnr = [(lat_fnr[i], long_fnr[i]) for i in range(len(lat_fnr))]
```

In [70]:

```
#Create map
this_map = folium.Map()

# Loop through 3 dataframes and plot point for each coordinate
for coord in lat_long_nf[::5]:
    folium.CircleMarker(location=[coord[0], coord[1]], opacity=0.6, color
for coord in lat_long_f[::5]:
    folium.CircleMarker(location=[coord[0], coord[1]], opacity=0.6, color
for coord in lat_long_fnr[::5]:
    folium.CircleMarker(location=[coord[0], coord[1]], opacity=0.6, color



#Set the zoom to fit our bounds
this_map.fit_bounds(this_map.get_bounds())

# Add legend
FloatImage('/Users/karaoglan/Desktoplegend.png', bottom=10, left=10).add_


this_map
```

Out[70]: Make this Notebook Trusted to load map: File -> Trust Notebook

As we saw above, there is a high rate of non functional waterpoints in the southeast corner of Tanzania in Mtwara and Lindi, as well as up north in Mara, and the southwest in Rukwa. We can see the cluster of high functional wells in Iringa, Shinyanga, Kilimanjaro, and Arusha. There is a cluster of functional but need repair waterpoints in Kigoma.

Create df['status'] with status_group in integer format

In [306…

```
# Change status_group/target values to numeric values
df['status'] = df.status_group.map({"non functional":0, "functional needs
df.head()
```

Out[306…

	status_group	amount_tsh	gps_height	installer	longitude	latitude	basin	r
0	functional	6000.0	1390	OTHER	34.938093	-9.856322	Lake Nyasa	
1	functional	0.0	1399	OTHER	34.698766	-2.147466	Lake Victoria	
2	functional	25.0	686	World Vision	37.460664	-3.821329	Pangani	Ma
3	non functional	0.0	263	OTHER	38.486161	-11.155298	Ruvuma / Southern Coast	M
4	functional	0.0	0	OTHER	31.130847	-1.825359	Lake Victoria	K

In [307…

```
df = df.drop('status_group', axis=1)
```

In [308…

```
df.shape
```

Out [308…

```
(53309, 18)
```

Modeling

Data Preprocessing

Following we will create our dummy variables for our categorical columns and perform train test split to prepare for modeling.

In [309…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53309 entries, 0 to 59399
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   amount_tsh             53309 non-null  float64
 1   gps_height             53309 non-null  int64  
 2   installer              53309 non-null  object 
 3   longitude              53309 non-null  float64
 4   latitude               53309 non-null  float64
 5   basin                  53309 non-null  object 
 6   region                 53309 non-null  object 
 7   population             53309 non-null  int64  
 8   permit                 53309 non-null  int64  
 9   construction_year      53309 non-null  int64  
 10  extraction_type_class  53309 non-null  object 
 11  management             53309 non-null  object 
 12  payment_type           53309 non-null  object 
 13  water_quality          53309 non-null  object 
 14  quantity               53309 non-null  object 
 15  source_type            53309 non-null  object 
 16  waterpoint_type        53309 non-null  object 
 17  status                 53309 non-null  int64  
dtypes: float64(3), int64(5), object(10)
memory usage: 7.7+ MB
```

One hot encoding

In [310…

```
from sklearn.preprocessing import OneHotEncoder
```

In [311…

```
# Create lists of categorical and continuous columns
cat_col = ['installer','basin','region','extraction_type_class','manageme
           'quantity','source_type','waterpoint_type']
cont_col = ['amount_tsh','gps_height','longitude','latitude','population'
```

In [312…

```
data = df.dropna(axis=0).reset_index(drop=True)
```

In [313…

```
df_cont = data[cont_col]
df_cat = data[cat_col]
```

In [314…

```
c = 0
for column in cat_col:
    print(column,"-->",len(data[column].unique()))
    c+= len(data[column].unique())
```

```
installer --> 21
basin --> 9
region --> 21
extraction_type_class --> 7
management --> 12
payment_type --> 7
water_quality --> 8
quantity --> 5
source_type --> 7
waterpoint_type --> 7
```

In [315…

```
enc = OneHotEncoder()
X_cat = enc.fit_transform(df_cat).toarray()
```

In [316…

```
X_cat = pd.DataFrame(X_cat, columns = enc.get_feature_names_out(cat_col))
```

In [317…

```
# 53309
X_cat = X_cat.reset_index(drop=True)
df_cont = df_cont.reset_index(drop=True)
```

In [318…

```
data_onehot = pd.concat([df_cont,X_cat], axis=1, ignore_index=True)
```

In [319…

```
data_onehot.columns =list(df_cont.columns) + list(X_cat.columns)
```

In [320…

```
# one_hot_encoded_data = pd.get_dummies(df, columns = ['installer','basi
#             'quantity','source_type','waterpoint_type']).head()
# print(one_hot_encoded_data.shape)
```

Separate target and perform train test split

In [321…

```
y = data_onehot['status']
X = data_onehot.drop(['status'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Model Statistics Function

Precision will be our main metric used to track model performance, but we will calculate accuracy, recall, and f1 score to provide more detail using sklearn's classificationreport() function _classificationreport() function.

In [322…

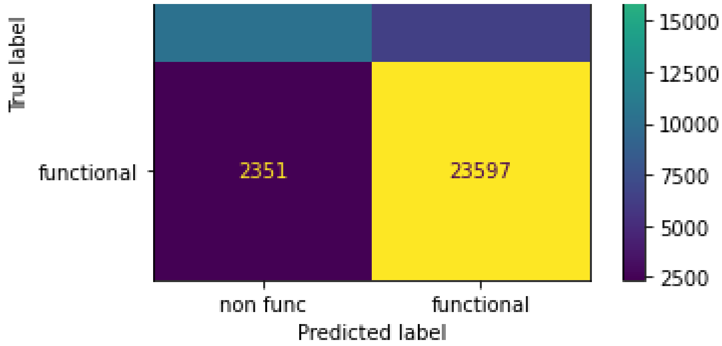
```
# function to track model metrics and plot confusion matrix

def model_score(model, X, y_pred, y_true):
#     target_names= ['non func', 'func need repair', 'functional']
    target_names= ['non func', 'functional']

    print(classification_report(y_true, y_pred, target_names=target_names

    #Confusion matrix
    return plot_confusion_matrix(model, X, y_true, display_labels=target_
```

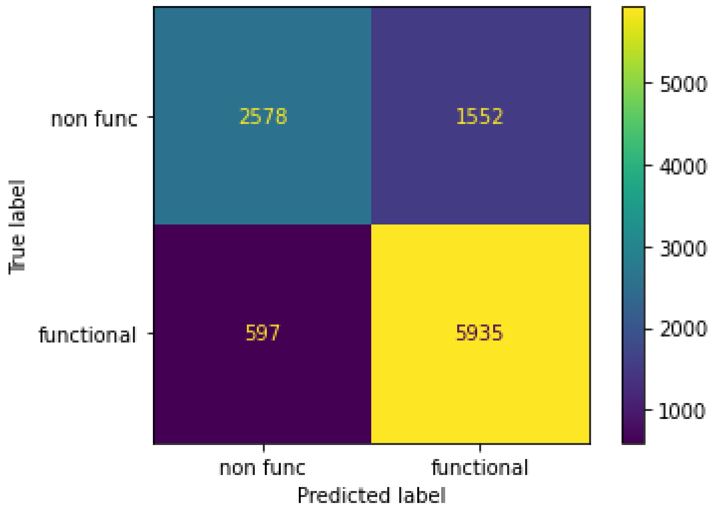
Logistic Regression

In [323…

```
# Make pipe
pipe_lr = Pipeline([('ss', StandardScaler()),
                    ('lr', LogisticRegression())])
#pipe_lr = Pipeline([
                    #('lr', LogisticRegression())])

# Fit and predict
pipe_lr.fit(X_train, y_train)
train_preds = pipe_lr.predict(X_train)
test_preds = pipe_lr.predict(X_test)


print("Test data model score:")

# test_preds
lr_score = model_score(pipe_lr, X_train, train_preds, y_train)
lr_score = model_score(pipe_lr, X_test, test_preds, y_test)
```

```
Test data model score:
              precision    recall  f1-score   support

    non func       0.81      0.62      0.70     16699
  functional       0.79      0.91      0.84     25948

    accuracy                           0.80     42647
   macro avg       0.80      0.76      0.77     42647
weighted avg       0.80      0.80      0.79     42647

              precision    recall  f1-score   support

    non func       0.81      0.62      0.71      4130
  functional       0.79      0.91      0.85      6532

    accuracy                           0.80     10662
   macro avg       0.80      0.77      0.78     10662
weighted avg       0.80      0.80      0.79     10662
```





In [ ]:

Our logistic regression model is improved to 75% accuracy over the dummy model. This model struggled to predict wells that were functional but needed repairs, likely due to class imbalances. The precision of the functional class is 73%.

K Nearest Neighbors

Below I will run GridSearch with my Pipeline to create a K Nearest Neighbors model. I ran GridSearch to find the best parameters, and have then commented out the code to save computing time while still showing the process. The same process is repeated for all following models of running GridSearch and commenting out code.

In [324…

```
# GridSearch
knn = KNeighborsClassifier()
grid = {
    'n_neighbors' : [5, 10, 15, 20, 25, 40]
}

knn_grid_search = GridSearchCV(knn, grid, cv=5)
knn_grid_search.fit(X_train, y_train)

knn_grid_search.best_params_
```

Out[324…

```
{'n_neighbors': 15}
```

In [325…

```
# Narrow down parameters for 2nd gridsearch
knn = KNeighborsClassifier()
grid = {

    'n_neighbors' : [ 11, 12, 13, 14, 15, 16, 17, 18]
}

knn_grid_search = GridSearchCV(knn, grid, cv=5)
knn_grid_search.fit(X_train, y_train)

knn_grid_search.best_params_
```

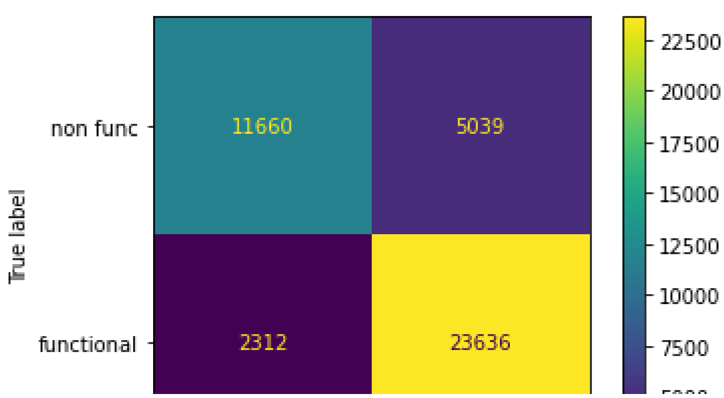
Out[325…

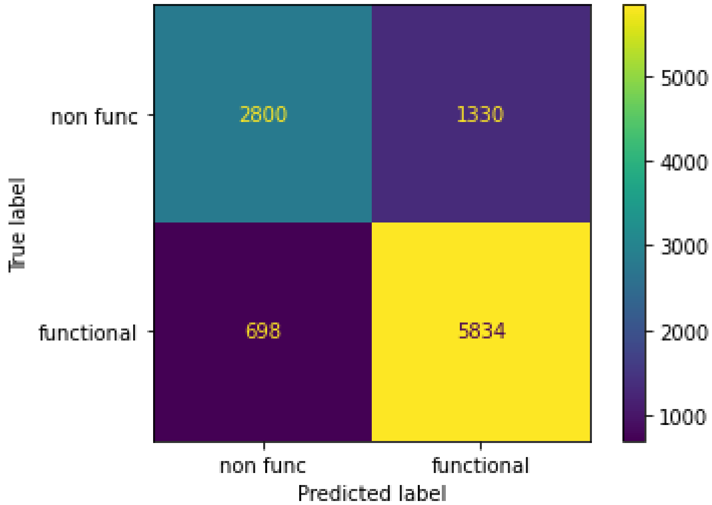
```
{'n_neighbors': 11}
```

In [327…

```
# Make pipe
pipe_knn = Pipeline([('ss', StandardScaler()),
                    ('knn', KNeighborsClassifier(n_neighbors=17))])
#Fit and predict
pipe_knn.fit(X_train, y_train)
train_preds = pipe_knn.predict(X_train)
test_preds = pipe_knn.predict(X_test)

print("Test data model score:")
knn_score = model_score(pipe_knn, X_train, train_preds, y_train)

knn_score = model_score(pipe_knn, X_test, test_preds, y_test)
```

```
Test data model score:
              precision    recall  f1-score   support

    non func       0.83      0.70      0.76     16699
  functional       0.82      0.91      0.87     25948

    accuracy                           0.83     42647
   macro avg       0.83      0.80      0.81     42647
weighted avg       0.83      0.83      0.82     42647

              precision    recall  f1-score   support

    non func       0.80      0.68      0.73      4130
  functional       0.81      0.89      0.85      6532

    accuracy                           0.81     10662
   macro avg       0.81      0.79      0.79     10662
weighted avg       0.81      0.81      0.81     10662
```





The K Nearest Neighbors model outperformed the Logistic Regression model. Number of neighbors was hypertuned by running and GridSearch and optimal parameters were put into our pipe. Our K Nearest Neighbors model is not overfitting as the accuracy of training and test sets are 80.23% and 76.03%, respectively. The precision of the functional class is 77%, which is a huge improvement from our Logistic Regression model at 73%.

Decision Tree Model

In [210…

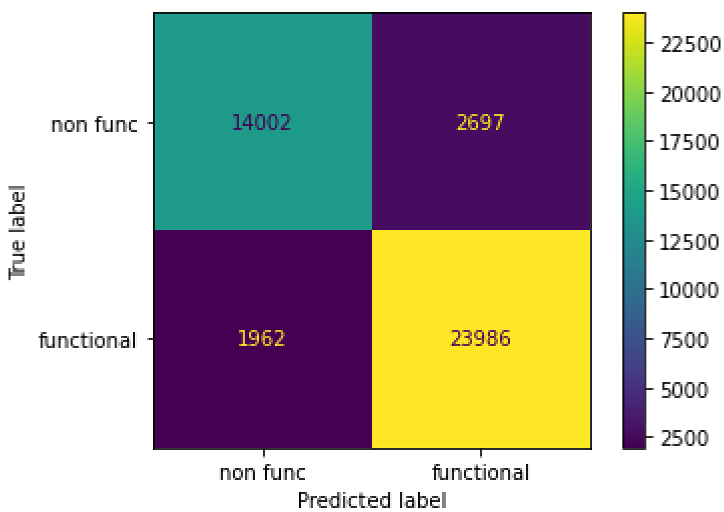
```
# GridSearch commented out to show process
# dt = DecisionTreeClassifier()
# dt_grid = {
#     'criterion' : ['entropy', 'gini'],
#     'max_depth': [10, 20, 30, 40, 50, 60, None],
#     'min_samples_split' : [1, 2, 5, 10, 20, 30],
#     'min_impurity_decrease' : [0.0, 0.1, 0.2, 0.3, 0.4, 0.5],
# #     'min_impurity_split' : [None, 0.1, 0.2, 0.3, 0.4, 0.5],
# }

# dt_tree = GridSearchCV(estimator=dt, param_grid=dt_grid, cv=5)
# dt_tree.fit(X_train, y_train)


# print(f'Best parameters are {dt_tree.best_params_}')
# print(f'Best score {dt_tree.best_score_}')  #0.768565112
# print(f'Best estimator score {dt_tree.best_estimator_.score(X_test, y_t

# "Best parameters are 'criterion': 'gini', 'max_depth': 30, 'min_impurit
#     'min_samples_split': 30", "Best score 0.8136796698169422,Best estim
```

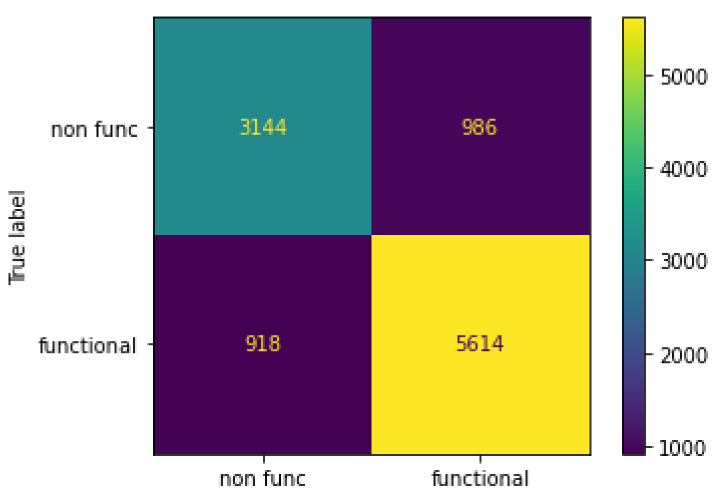
In [211…

```
# Make pipe
pipe_dt = Pipeline([('ss', StandardScaler()),
                    ('dt', DecisionTreeClassifier(criterion='gini', max_d
```

```
                                        min_samples_split=30))]
#Fit and Predict
pipe_dt.fit(X_train, y_train)
test_preds = pipe_dt.predict(X_test)
train_preds = pipe_dt.predict(X_train)

print("Test data model score:")
dt_score = model_score(pipe_dt, X_train, train_preds, y_train)
dt_score = model_score(pipe_dt, X_test, test_preds, y_test)
```

```
Test data model score:
              precision    recall  f1-score   support

    non func       0.88      0.84      0.86     16699
  functional       0.90      0.92      0.91     25948

    accuracy                           0.89     42647
   macro avg       0.89      0.88      0.88     42647
weighted avg       0.89      0.89      0.89     42647

              precision    recall  f1-score   support

    non func       0.77      0.76      0.77      4130
  functional       0.85      0.86      0.86      6532

    accuracy                           0.82     10662
   macro avg       0.81      0.81      0.81     10662
weighted avg       0.82      0.82      0.82     10662
```





Predicted label

Function to plot feature importances

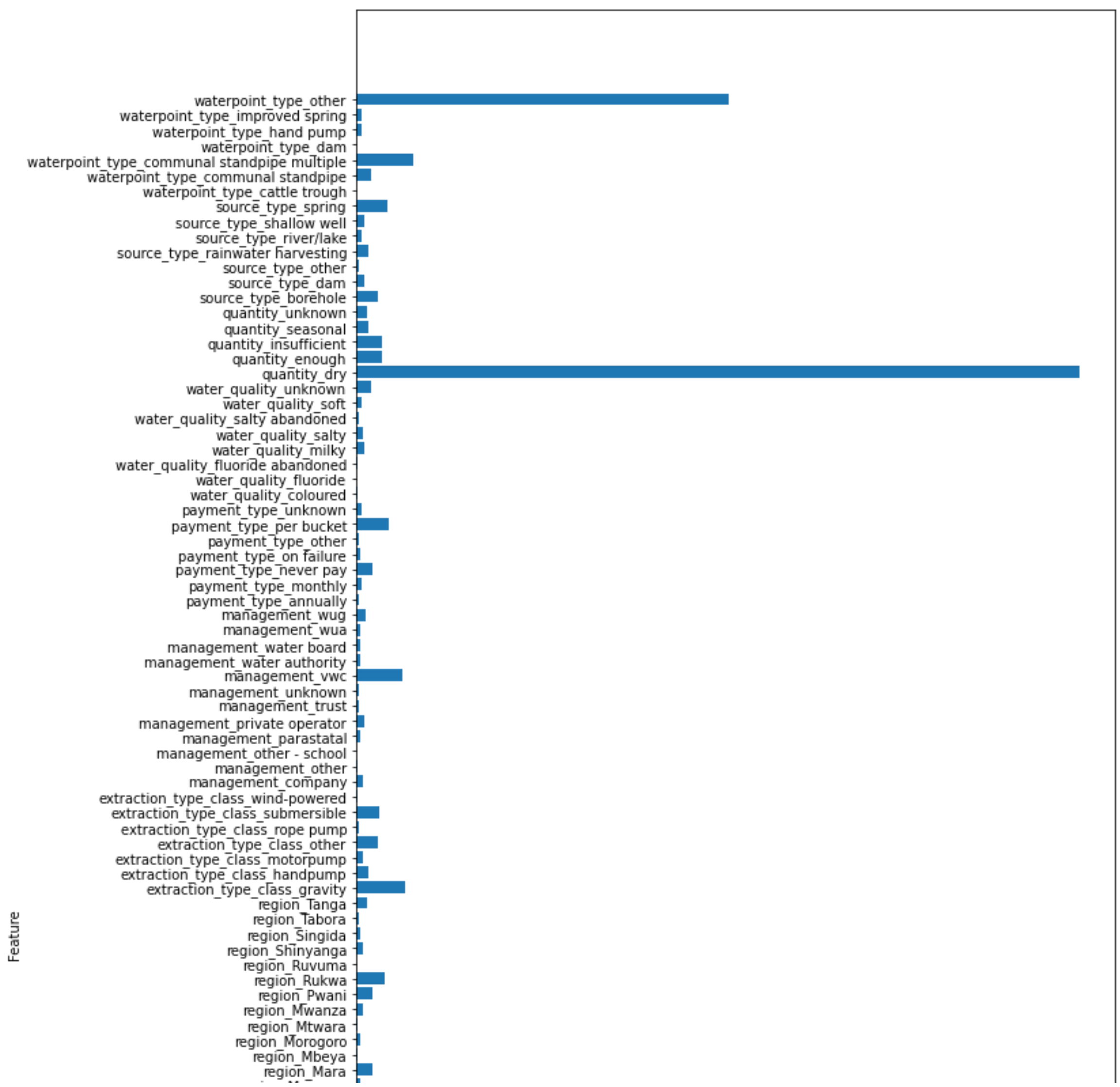
In [232…

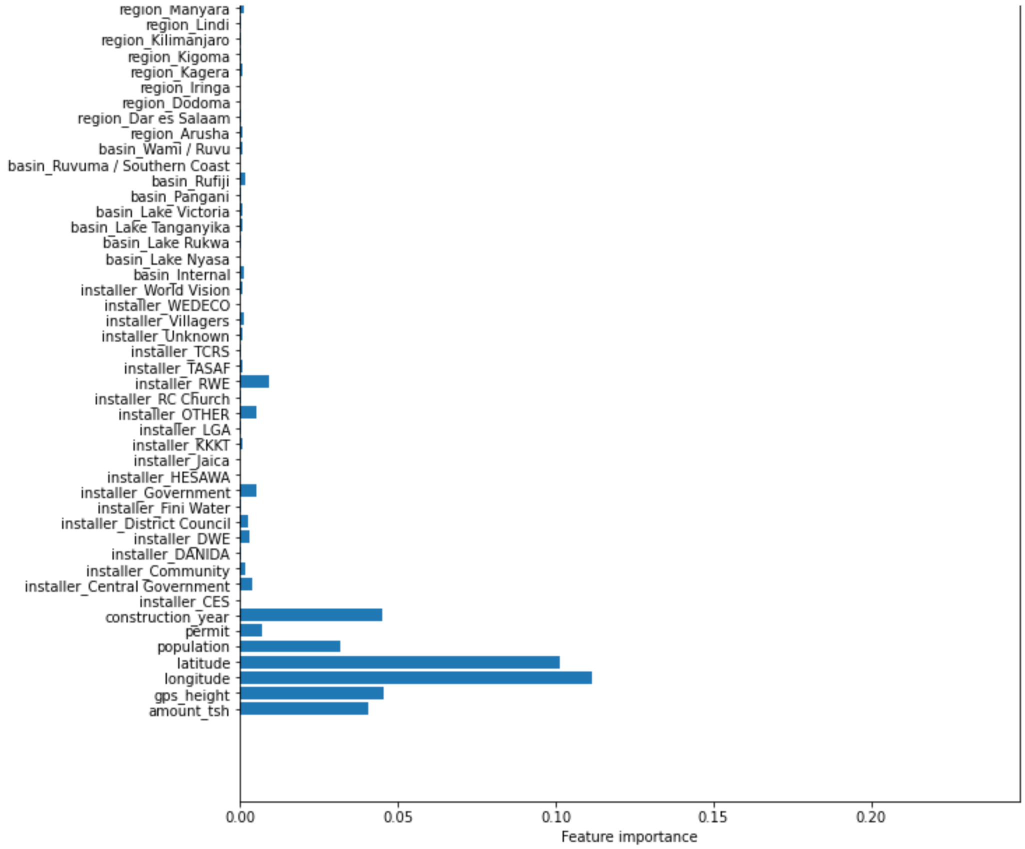
```
def plot_feature_importances(model):
    n_features = X_train.shape[1]
    plt.figure(figsize=(10,25))
    plt.barh(range(n_features), model.feature_importances_, align='center
    plt.yticks(np.arange(n_features), X_train.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')
```

Decision Tree Feature Importances

In [233…

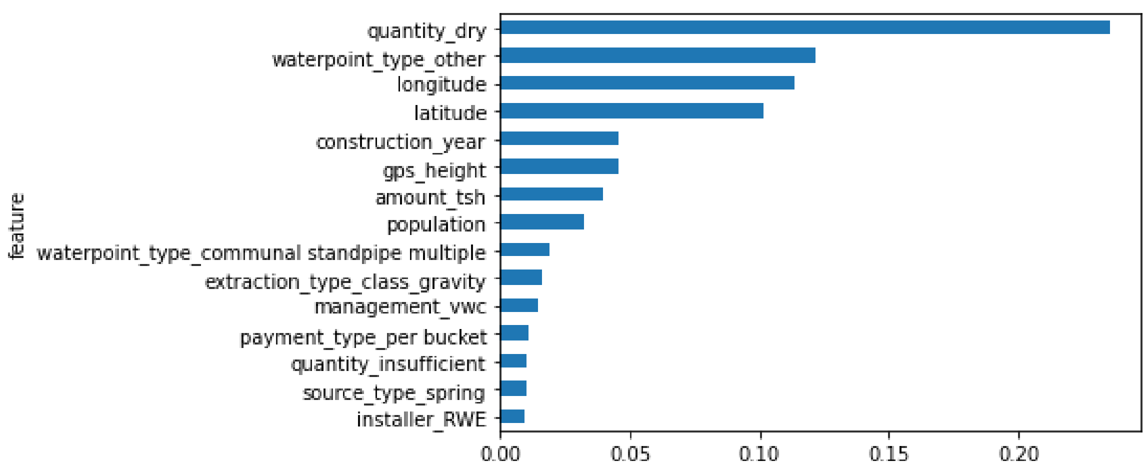
```
# Instantiate and fit a DecisionTreeClassifier with optimal parameters
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=30, min_imp
tree_clf.fit(X_train, y_train)

plot_feature_importances(tree_clf)
```





In [214…

```python
# Top features
feature_importances=pd.DataFrame(columns=['feature','importance'])

feature_importances['feature']= X_train.columns

feature_importances['importance']=tree_clf.feature_importances_

feature_importances= feature_importances.set_index('feature')

feature_importances['importance'].sort_values(ascending = True).tail(15).
```



Our Decision Tree Feature Importances model shows the most important features to be

- quanity_dry
- waterpoint_type_other

- longitude
- latitude

Random Forests

In [234…

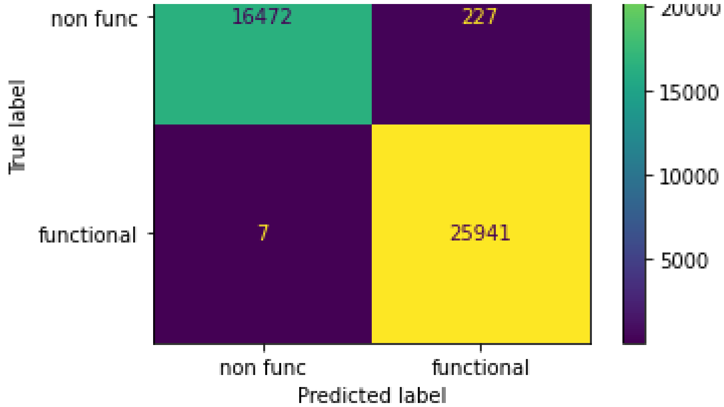
```
#Instantiate RandomForestClassifier
forest = RandomForestClassifier(n_estimators=100, max_depth= 5)
forest.fit(X_train, y_train)

#scores on folds
scores = cross_val_score(estimator=forest, X=X_train, y=y_train, cv=5)
print(np.mean(scores))

#scores on on test
score = forest.score(X_test, y_test)
print(score)
```

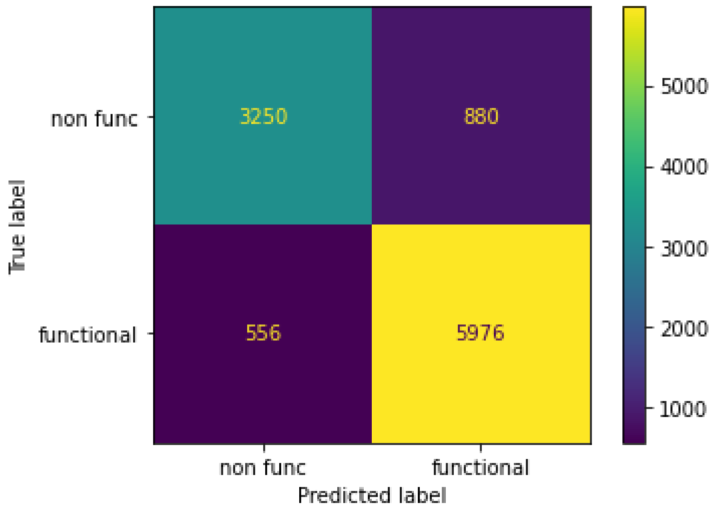
```
0.7725982782416573
0.7784655786906771
```

In [235…

```
# Make pipeline with tuned hyperparameters
pipe_rf = Pipeline([('ss', StandardScaler()),
                    ('RF', RandomForestClassifier(bootstrap=True, criteri
# Fit and predict
pipe_rf.fit(X_train, y_train)

pipe_rf.fit(X_train, y_train)
test_preds = pipe_rf.predict(X_test)

# Print metrics
print("Test data model score:")
rf_score = model_score(pipe_rf, X_train, train_preds, y_train)
rf_score = model_score(pipe_rf, X_test, test_preds, y_test)
```

```
Test data model score:
              precision    recall  f1-score   support

    non func       0.88      0.84      0.86     16699
  functional       0.90      0.92      0.91     25948

    accuracy                           0.89     42647
   macro avg       0.89      0.88      0.88     42647
weighted avg       0.89      0.89      0.89     42647

              precision    recall  f1-score   support

    non func       0.85      0.79      0.82      4130
  functional       0.87      0.91      0.89      6532

    accuracy                           0.87     10662
   macro avg       0.86      0.85      0.86     10662
weighted avg       0.86      0.87      0.86     10662
```





In [216…

```
rf_param_grid = {
    'n_estimators': [10, 30, 100],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 6, 10],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [3, 6]
}
```

In [217…

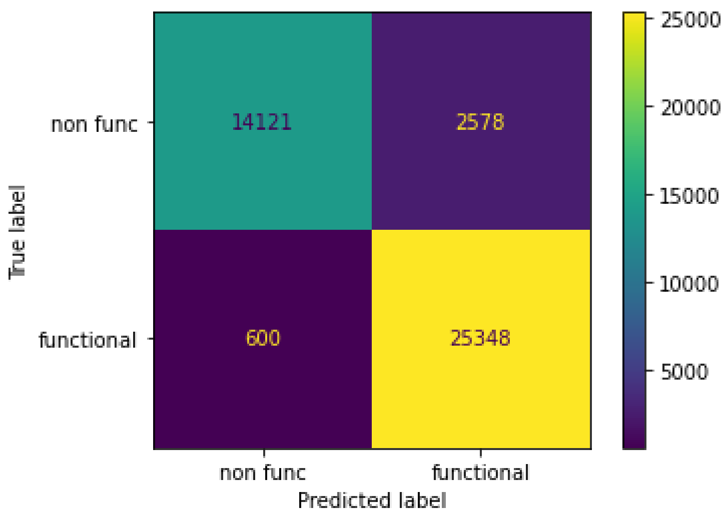
```
# rf_grid_search = GridSearchCV(rf_clf, rf_param_grid, cv=5)
# rf_grid_search.fit(X_train, y_train)

# print(f"Training Accuracy: {rf_grid_search.best_score_ :.2%}")
# print("")
# print(f"Optimal Parameters: {rf_grid_search.best_params_}")
# Training Accuracy: 84.48%

# Optimal Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_sa
```

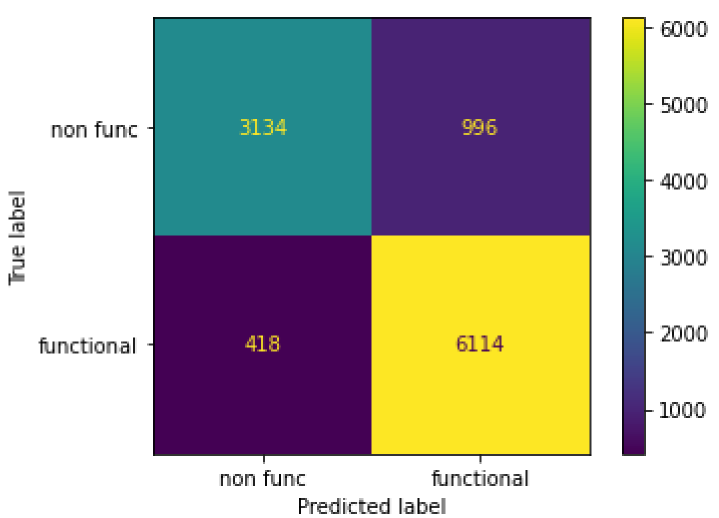
In [330…

```
# Make pipeline with tuned hyperparameters
pipe_rf = Pipeline([('ss', StandardScaler()),
                    ('RF', RandomForestClassifier(bootstrap=True, criteri
                                                  min_samples_leaf= 2, mi
# Fit and predict
pipe_rf.fit(X_train, y_train)
test_preds = pipe_rf.predict(X_test)

# Print metrics
```

```
print("Test data model score:")
rf_score = model_score(pipe_rf, X_train, train_preds, y_train)

rf_score = model_score(pipe_rf, X_test, test_preds, y_test)
```

```
Test data model score:
              precision    recall  f1-score   support

    non func       0.83      0.70      0.76     16699
  functional       0.82      0.91      0.87     25948

    accuracy                           0.83     42647
   macro avg       0.83      0.80      0.81     42647
weighted avg       0.83      0.83      0.82     42647

              precision    recall  f1-score   support

    non func       0.88      0.76      0.82      4130
  functional       0.86      0.94      0.90      6532

    accuracy                           0.87     10662
   macro avg       0.87      0.85      0.86     10662
weighted avg       0.87      0.87      0.87     10662
```



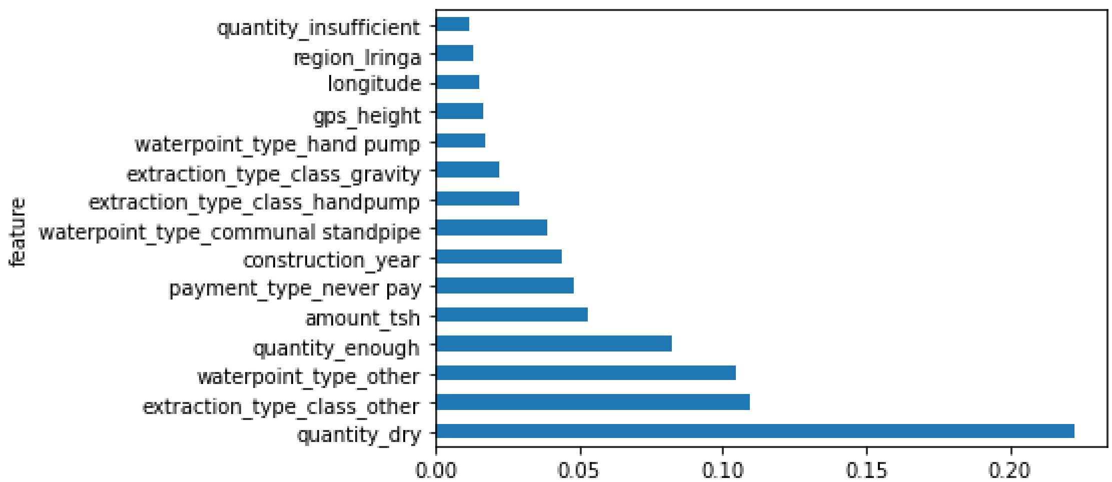


In [ ]:

Random Forests Feature Importances

In [341…

```
# Top features
feature_importances=pd.DataFrame(columns=['feature','importance'])

feature_importances['feature']= X_train.columns

feature_importances['importance']=forest.feature_importances_

feature_importances= feature_importances.set_index('feature')

feature_importances['importance'].sort_values(ascending = False).head(15)
```



Our Random Forests model shows the most important features to be

- quanity_dr
- waterpoint_type_other
- extraction_type_class_other
- quantity_enough

In [238…

```
forest.feature_importances_
```

Out[238…

```
array([5.28540108e-02, 1.62855093e-02, 1.48722445e-02, 1.03141461e-02,
       9.07013951e-03, 1.35930342e-03, 4.38421928e-02, 2.56706343e-04,
       4.61644425e-03, 6.05110763e-04, 1.05473964e-04, 5.37782964e-04,
       5.67146451e-04, 4.40832657e-03, 1.03993248e-03, 8.28624326e-05,
       5.53294783e-05, 3.37491002e-04, 9.02542485e-05, 1.02384908e-03,
       2.39786971e-05, 1.60293098e-03, 3.17395249e-06, 8.56772453e-05,
       2.32876684e-04, 7.55404956e-05, 1.53952559e-04, 3.79393312e-05,
       1.50304635e-03, 2.75492929e-03, 1.83664291e-04, 3.39730684e-04,
       2.36434115e-03, 1.08321665e-03, 6.39299034e-04, 2.83945830e-03,
       5.33976782e-04, 8.69480336e-04, 2.25023557e-05, 8.47568247e-04,
       1.33903111e-02, 9.20522417e-04, 6.52031110e-04, 5.54769635e-04,
       3.70944981e-04, 5.61339811e-05, 2.67167008e-03, 1.29376402e-03,
       2.29855339e-04, 2.13484092e-03, 5.51999710e-04, 4.22942510e-04,
       1.26961143e-03, 3.42386438e-04, 2.05868243e-03, 4.32930860e-04,
       5.22217296e-05, 1.73001007e-04, 2.23011929e-02, 2.89345995e-02,
       3.75868843e-03, 1.09528077e-01, 4.90641187e-05, 3.45899871e-03,
       2.27445842e-05, 3.13664439e-03, 1.51272183e-04, 3.81153508e-05,
```

```
       1.35148318e-04, 1.77686245e-03, 4.30586113e-05, 2.41821419e-04,
       1.15487394e-02, 2.37931167e-04, 8.09498654e-03, 1.22121842e-03,
       1.35845190e-03, 4.15535645e-03, 1.03866835e-02, 4.78622623e-02,
       4.69326334e-04, 2.77238656e-05, 3.83580120e-03, 5.77490159e-03,
       1.74333556e-05, 1.06230208e-04, 0.00000000e+00, 4.68119684e-04,
       2.76210617e-04, 5.55531886e-06, 3.44426674e-03, 8.10265963e-03,
       2.22621114e-01, 8.23188242e-02, 1.16959501e-02, 4.86603723e-03,
       1.15334872e-03, 4.81498332e-03, 2.67821250e-04, 2.56881034e-05,
       2.57954358e-03, 5.27805380e-04, 7.02192257e-03, 4.41713642e-03,
       8.11062097e-06, 3.92002555e-02, 7.82208476e-03, 0.00000000e+00,
       1.71151597e-02, 1.97087003e-03, 1.04503039e-01])
```

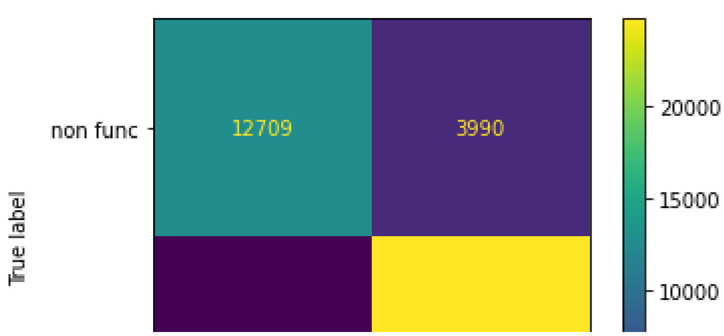
Our random forests model show waterpoint_type other, enough quantity, extraction_type_class_other, and amount_tsh being the most important features to the model.

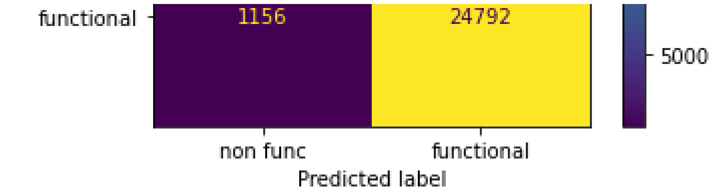
XG Boost

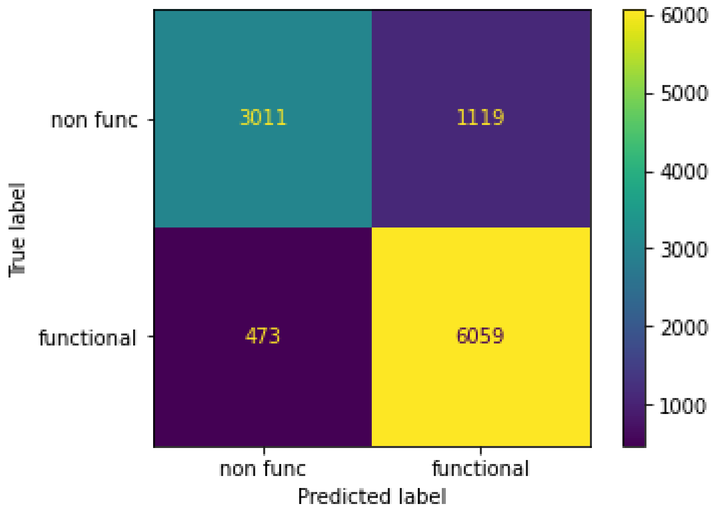
In [328…

```python
# Instantiate XGBClassifier
xgb = XGBClassifier()

# Fit XGBClassifier
xgb.fit(X_train, y_train)

print("Test data model score:")
xgb_model_score = model_score(xgb, X_train, train_preds, y_train)
xgb_model_score = model_score(xgb, X_test, test_preds, y_test)
```

```
Test data model score:
              precision    recall  f1-score   support

    non func       0.83      0.70      0.76     16699
  functional       0.82      0.91      0.87     25948

    accuracy                           0.83     42647
   macro avg       0.83      0.80      0.81     42647
weighted avg       0.83      0.83      0.82     42647

              precision    recall  f1-score   support

    non func       0.80      0.68      0.73      4130
  functional       0.81      0.89      0.85      6532

    accuracy                           0.81     10662
   macro avg       0.81      0.79      0.79     10662
weighted avg       0.81      0.81      0.81     10662
```







In [ ]:

```
# # #Gridsearch commented out
# xgb = XGBClassifier()
# grid = {
#      'learning_rate': [0.01,0.05,0.1],
#     'max_depth': [5,10,15],
#      'subsample': [0.5, 0.7,0.9],
#      'n_estimators': [100,500,1000]

#   }

# gs_xgb = GridSearchCV(estimator=xgb, param_grid=grid, cv=5)
# gs_xgb.fit(X_train, y_train)

# # print(f'Best parameters are {gs_xgb.best_params_}')
# # print(f'Best score {gs_xgb.best_score_}')
# # print(f'Best estimator score {gs_xgb.best_estimator_.score(X_test, y_
```

In [ ]:

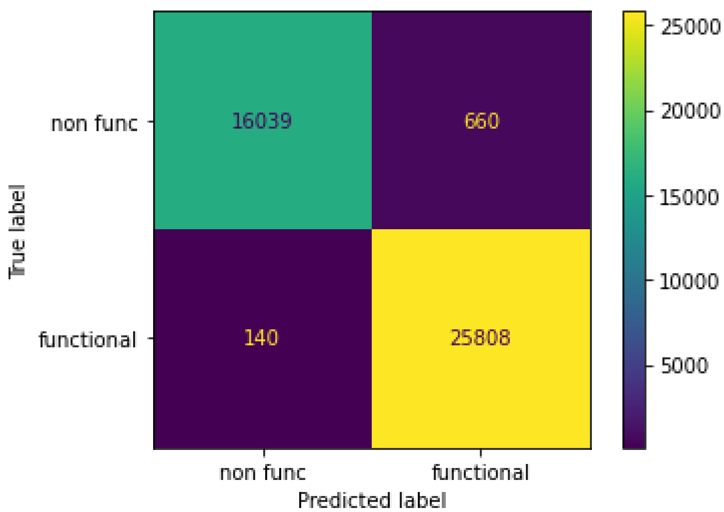
```
 gs_xgb.best_params_
    {'learning_rate': 0.01,
  'max_depth': 10,
  'n_estimators': 1000,
  'subsample': 0.9}
```

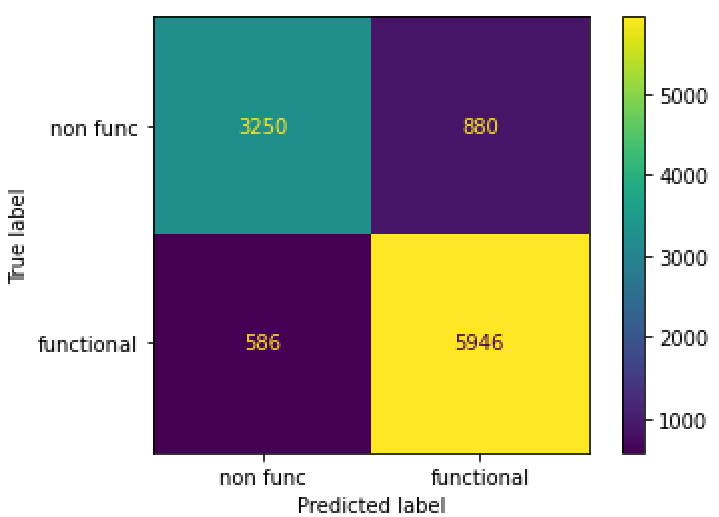
In [329…

```
# Instantiate xg Boost Classifier pipeline with tuned hyperparameters
pipe_xgb = Pipeline([('ss', StandardScaler()),
                     ('xgb', XGBClassifier(learning_rate=0.1, max_depth=1
                      n_estimators=1000, subsample=0.9))])

pipe_xgb.fit(X_train, y_train)
test_preds = pipe_xgb.predict(X_test)

print("Test data model score:")
dt_score = model_score(pipe_xgb, X_train, train_preds, y_train)
```

```
dt_score = model_score(pipe_xgb, X_test, test_preds, y_test)
```

```
Test data model score:
              precision    recall  f1-score   support

    non func       0.83      0.70      0.76     16699
  functional       0.82      0.91      0.87     25948

    accuracy                           0.83     42647
   macro avg       0.83      0.80      0.81     42647
weighted avg       0.83      0.83      0.82     42647

              precision    recall  f1-score   support

    non func       0.85      0.79      0.82      4130
  functional       0.87      0.91      0.89      6532

    accuracy                           0.86     10662
   macro avg       0.86      0.85      0.85     10662
weighted avg       0.86      0.86      0.86     10662
```





In [ ]:

```
# # Predict on training and test sets
# training_preds = pipe_xgb.predict(X_train)
# test_preds = pipe_xgb.predict(X_test)

# # Accuracy of training and test sets
```

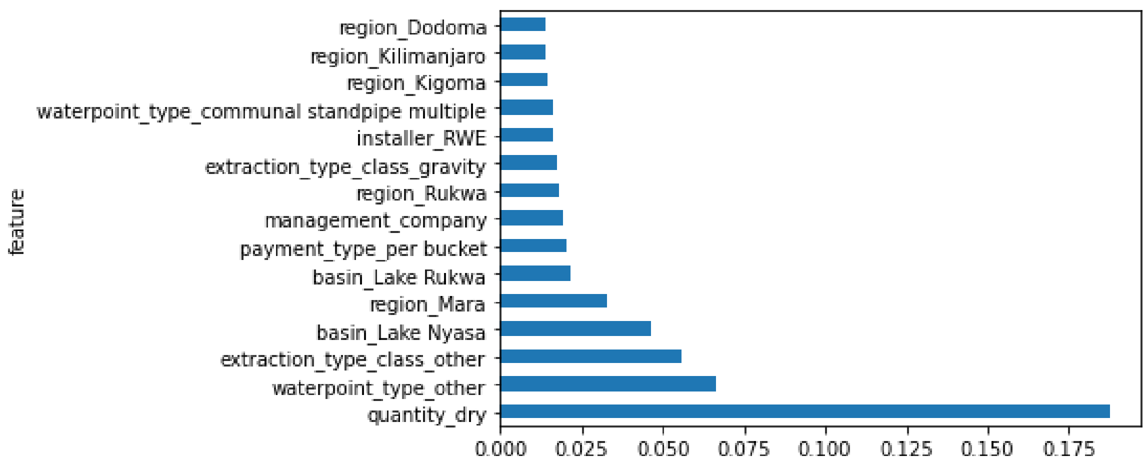
```
# training_accuracy = accuracy_score(y_train, training_preds)
# test_accuracy = accuracy_score(y_test, test_preds)

# print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
# print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

Our best performing model ended up being the XG Boost model with tuned hyperparameters, although the random forests model was not far behind with 80% precision for the functional wells class. The model has overfitted the training data with a training accuracy of 92.57% and test accuracy at 81.73%, but this model boasted the highest precision score for the functional wells class at 81%.

XGB Feature Importances

In [338…

```
feature_importances=pd.DataFrame(columns=['feature','importance'])
feature_importances['feature']= X_train.columns
feature_importances['importance']=xgb.feature_importances_
feature_importances= feature_importances.set_index('feature')
feature_importances['importance'].sort_values(ascending = False).head(15)
```



Our XG Boost model shows the most important features to be

- quantity_dry,
- waterpoint_type_class_other,
- extraction_type_class_other,
- managment_company

Bussines Problem

- The Tanzanian goverment has a severe water crisis on their hands
- They want to predict which pumps are functional, functional but need repairs, and non functional
- Taarifa and Tanzanian Ministry of Water have shared the dataset to aid

- Taarifa and Tanzanian Ministry of Water have shared the dataset to aid understanding of pump failure
- I will build model to help the goverment improve maintenance operations
- And ensure clean drinking waer is accessible to communities acrosstanzania

Recommendations

- Location
  - Target repairs in areas like Lindi and Mtwara that have a high rateof non functional wells
  - Make repairs to functional wells in Kignma to maximize cost effectivess
- Repairs
  - Prioritize non functional and functional wells which need repair and have enough water
- Payment
  - Payment provides incentive and means to keep ells functional
- Installers
  - Avoid using installers with a high rate of pump failure

Conclusions

Random Forests was our top performing model, although XG Boost was not far behind. The poor performance of the Logistic Regression , KNN, and Decision Tree indicate that the data is not easily separable. Our Random Forests model performs with an 87% testing accuracy and precision for the functional class at 86%.

Several of our models showed one of it's most important features to be quantity for the waterpoint. There are over 8,000 waterpoints that have enough water in them but are non functional. These are a high priority to address as well since there is water present. Wells with no fees are more likely to be non functional. Payment provides incentive and means to keep wells functional. The Government, District Council, and Fini Water all have a high rate of pump failure. Investigate why these installers have such a high rate of failure or use other installers.

Decision Tree Feature Importances model shows the most important features to be

- quantity_dry
- waterpoint_type_other
- longitude
- latitude

Our Random Forests model shows the most important features to be

- quantity_dr
- waterpoint_type_other
- extraction_type_class_other
- quantity_enough

Our XG Boost model shows the most important features to be

- quantity_dry,
- waterpoint_type_class_other,
- extraction_type_class_other,
- managment_company

Future work

Future work for this project involve improving the quality of the data moving forward. Better data trained in our model will improve the predictions. We will also monitor the wells and update the model regularly to continuously improve our strategy.

In [ ]: